21世纪高等学校计算机类课程创新规划教材 · 微课版

Python程序设计案例教程

从入门到机器学习

微课版

张思民 编著



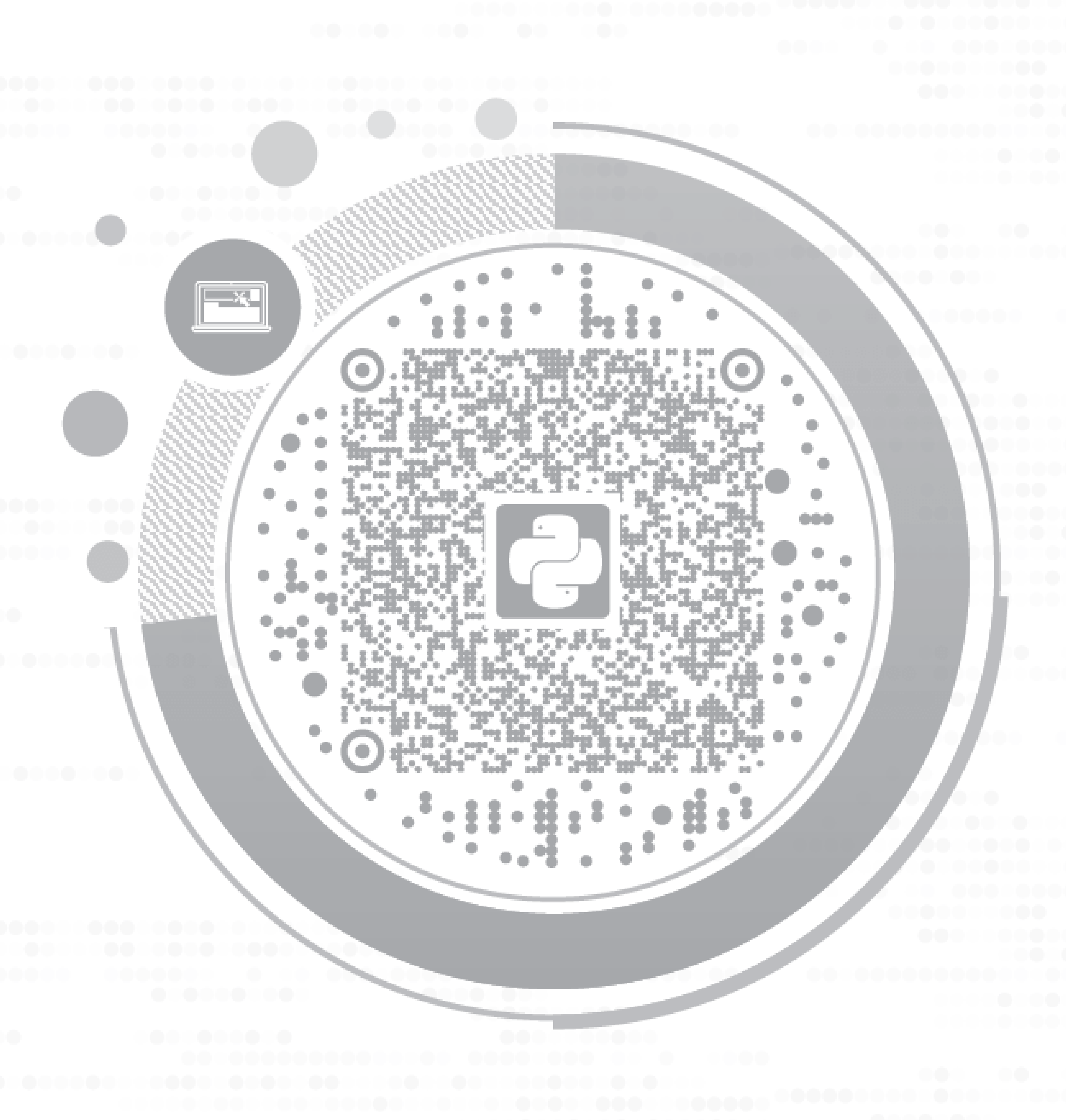




- 语法快速入门
- 类与模块
- 多线程
- 绘图及数字图像处理
- 文件与数据库操作
- 图形用户界面设计
- 异常处理及正则表达式
- 网络编程与网络爬虫设计
- 机器学习

清华大学出版社

21世纪高等学校计算机类课程创新规划教材 · 微课版



Python程序设计案例教程

从入门到机器学习

微课版

张思民 编著

清華大学出版社

北京

内 容 简 介

本书系统地介绍 Python 应用程序设计方法，主要内容包括 Python 语法快速入门、类与模块、图形用户界面设计、绘图及数字图像处理、文件与数据库操作、多线程、异常处理及正则表达式、网络编程与网络爬虫设计、算法设计与机器学习实战等。本书每章都配有视频教学内容，以帮助读者学习和理解。

本书讲解详细，案例丰富，每一个知识点都配备了大量案例和图示加以说明，并通过典型案例对 Python 应用程序设计方法进行详细的分析和解释，帮助读者轻松上手。

图书在版编目（CIP）数据

Python 程序设计案例教程：从入门到机器学习：微课版/张思民编著. —北京：清华大学出版社，2018
（21 世纪高等学校计算机类课程创新规划教材·微课版）
ISBN 978-7-302-51014-7

Ⅰ. ①P…　Ⅱ. ①张…　Ⅲ. ①软件工具 – 程序设计 – 高等学校 – 教材　Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2018）第 186984 号

责任编辑： 魏江江　赵晓宁
封面设计： 刘　键
责任校对： 李建庄
责任印制： 丛怀宇

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn, http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn，010-62795954
印 装 者： 北京鑫海金澳胶印有限公司
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 16　　**字　　数：** 388 千字
版　　次： 2018 年 10 月第 1 版　　**印　　次：** 2018 年 10 月第 1 次印刷
印　　数： 1～1500
定　　价： 49.00 元

产品编号：080369-01

前　言

Python 是一种面向对象的解释型计算机程序设计语言。这门强大的语言如今在大学和一些大型软件开发公司中广泛使用，其应用也越来越广。

本书从 Python 初学者的角度进行选材和编写，在编写过程中，注重基础知识和实战应用相结合，本书有以下几个特点：

（1）浅显易懂。本书从人们认知规律出发，对每一个概念，用简单的示例或图示来加以说明，并用短小的典型示例进行分析解释。

（2）内容新颖而实用。人们学习编程的目的是为了解决人们生活和生产实践中的问题，本书使用 Python 3.x 以上版本编写代码，大部分章节精选了实用案例，可以帮助解决读者在学习和实际应用过程中所遇到的一些困难和问题。

（3）本书在体系结构的安排上将 Python 编程的基础知识和一般编程思想有机结合，对基础知识重点介绍与其他编程语言不同的部分，而与其他编程语言相同的语法部分则简略介绍。因此，本书适合具有初步编程语言基础的读者学习。

本书共 9 章，其内容简单介绍如下。

第 1 章主要介绍 Python 的安装与配置、Python 程序编写规范和简单的 Python 程序示例。

第 2 章简要地介绍数据类型、列表与元组、字典与集合、程序的三大控制结构（顺序结构、分支结构、循环结构）、函数的基本语法与应用。

第 3 章主要介绍类与模块的基本知识，并介绍了使用 pip 安装和管理扩展模块的方法。

第 4 章主要介绍窗体容器、按钮和文本框等组件、界面布局管理等图形用户界面设计的方法，还介绍了鼠标与键盘事件及其应用示例。

第 5 章主要介绍绘图与数字图像处理的基本方法。

第 6 章主要介绍数据的存储，包括文件的读写、对 Excel 表格的处理、对 SQLite 数据库及 MySQL 数据库记录增删改查的操作。

第 7 章主要介绍多线程、异常处理及正则表达式。

第 8 章主要介绍基于 TCP 及 UDP 的套接字编程和网络爬虫程序的设计，并介绍了爬取网络数据的几个典型案例，还介绍了 Python 在网络程序开发中的方法和技巧，旨在提升读者的开发技能，达成学以致用之目标。

第 9 章主要介绍了常见数据结构，还介绍了两个 Python 的热门算法设计应用——数据分析和机器学习的应用案例。

建议教学安排（根据课程设置了两个课时分配方案）：

章　节	方案 1/学时	方案 2/学时
第 1 章　Python 语言快速入门	2	2
第 2 章　Python 语法速览	4	8
第 3 章　类与模块	2	4

续表

章　　节	方案 1/学时	方案 2/学时
第 4 章　图形用户界面设计	4	8
第 5 章　绘图与图像处理	4	6
第 6 章　文件与数据库编程（数据存储）	6	12
第 7 章　多线程与异常处理	2	6
第 8 章　网络程序设计	6	14
第 9 章　算法设计及机器学习实战入门	2	4
合计	32	64

学编程必须动手才能见到成效，本书在设计上特别强调讲练结合，注重实践，不仅在讲解的过程中结合大量代码示例，同时适时穿插小项目演练，以锻炼读者的程序设计能力。

有很多人认为 Python 简单易学，但其实 Python 的复杂程度要远高于许多人的想象，诸多概念被隐藏在看似简单的代码背后。这也是 Python 易学难精的主要原因。因此，要强调动手实践，多编写、多练习，熟能生巧，从学习中体验到程序设计的乐趣和成功的喜悦，增强学习信心。

本书由张思民编著。梁维娜参加本书编写及程序测试工作，在此表示感谢。

编　者

2018 年 5 月

目　录

第 1 章　Python 语言快速入门

Python 是一种面向对象的解释型计算机编程语言。Python 语言具有通用性、高效性、跨平台移植性和安全性，广泛应用于科学计算、自然语言处理、图形图像处理、游戏开发、Web 应用等方面，在全球范围内拥有众多开发者专业社群。

视频录像

1.1　Python 的安装与配置

1.　Python 的下载和安装

学习 Python 需要一个程序开发环境。只有安装并配置了 Python 系统开发环境之后，Python 程序才能运行。经过长期发展，Python 同时流行两个不同的版本，它们分别是 2.7.x 和 3.x 版本。注意，这两个版本是不兼容的，本书是基于 3.x 版本编写的。

可以在 Python 的官方网站 https://www.python.org/downloads/下载 Python 安装包，如图 1.1 所示。



图 1.1　Python 安装包下载

下载 Python 安装包后，就可以运行安装程序，进入 Python 的安装界面，按照提示完成安装。

2. Python 开发环境的配置

安装完成后，还需要配置 Python 的环境变量。这里假设 Python 系统安装在“C:\Python\”目录下。

在 Windows 操作系统下，右击桌面上的“计算机”图标，在弹出的快捷菜单中选择“属性”选项，打开“系统属性”对话框。在“系统属性”对话框的“高级”选项卡中，单击“环境变量”按钮，打开“环境变量”对话框。在“系统变量”列表框中，双击 Path 变量，打开“编辑系统变量”对话框。在“变量值”文本框中输入填写 Python 的安装路径，这里输入“C:\Python\”，如图 1.2 所示。

配置完成后，不论当前目录是在何处，执行 Python 命令时，操作系统都会执行这条命令，从而可以在任何目录下执行 Python 源程序代码。

3. Python 在线帮助文档

Python 还提供非常完善的 Python 帮助文档，这是进行程序设计的工具。Python 帮助文档在 Python 安装目录的 doc 目录下，双击即可打开，如图 1.3 所示。



图 1.2　设置环境变量

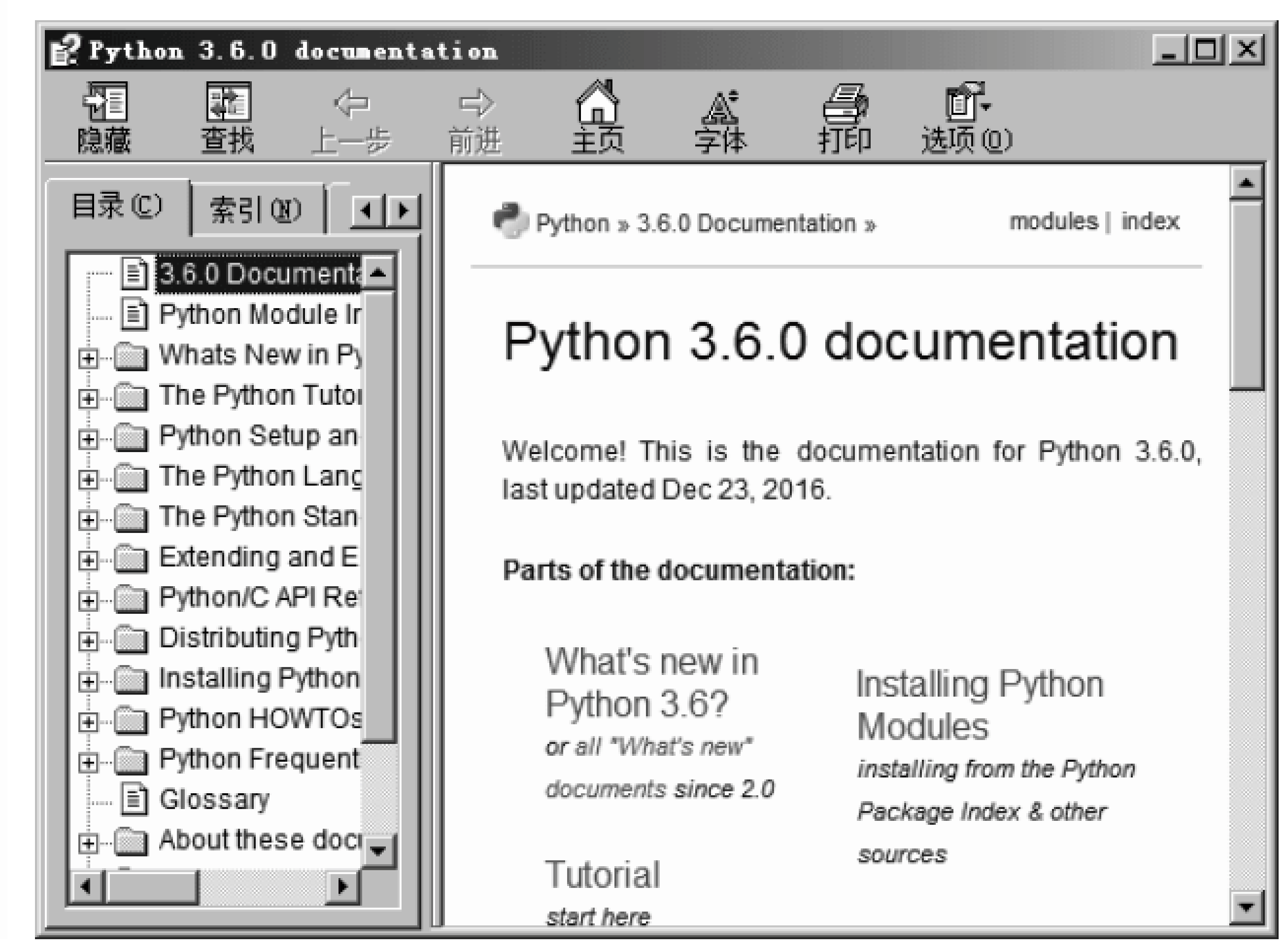


图 1.3　Python 在线帮助文档

1.2　运行 Python 程序

1.2.1　运行 Python 的方式

运行 Python 有两种方式：一种是命令行的交互方式；另一种是使用源程序文件方式。

1. 命令行交互方式

选择“开始”→“所有程序”→Python→IDLE 菜单项，启动 Python 运行环境，进入交互编程方式。

在 IDLE 提示符“>>>”后面输入单条 Python 语句，按 Enter 键执行该语句，马上就可

以看到执行结果，如图 1.4 所示。

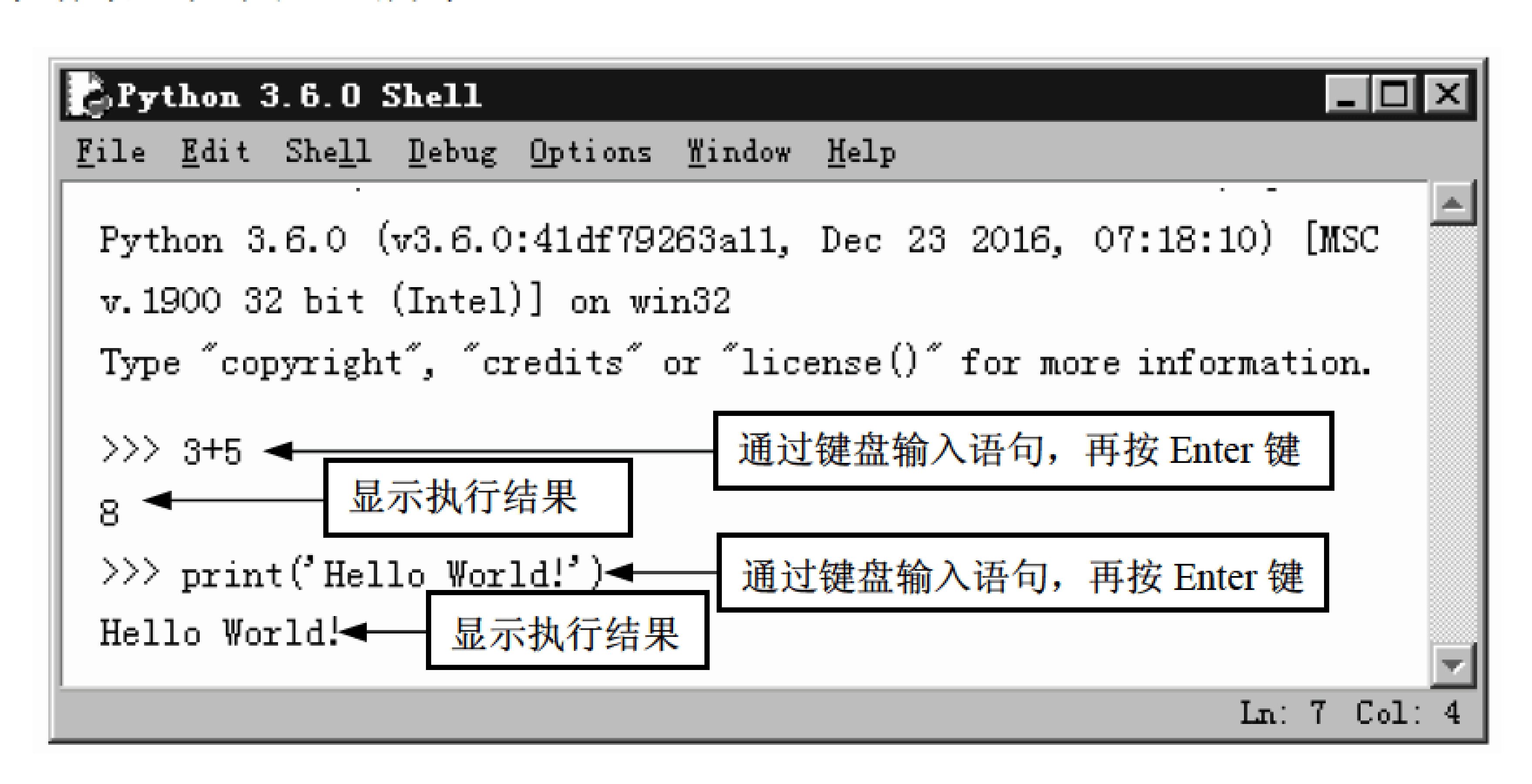


图 1.4　Python 交互方式

2. 源程序的文件方式

Python 应用程序的开发方式：使用文本编辑器，编写 Python 源程序，并保存扩展名为 py 的文件。

Python 应用程序的开发过程如图 1.5 所示。

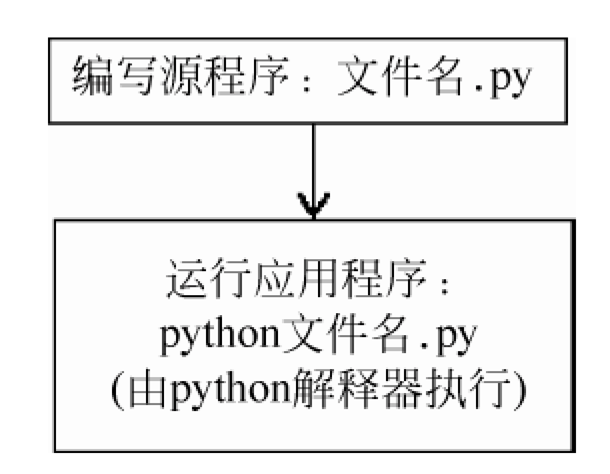


图 1.5　Python 程序的开发过程

（1）建立 Python 源文件

要建立一个 Python 程序，首先创建 Python 的源代码，即建立一个文本文档，包括有符合 Python 规范的语句。

开发一个 Python 程序必须遵循如下基本原则：

- Python 程序中一行就是一条语句，语句结束不需要使用分号；
- Python 采用缩进格式标记一组语句，缩进量相同的是同一组语句，也称为程序段；
- 一条语句也可以分多行书写，用反斜杠（\）表示续行。

例如：

```
a = (3 + 2) * (6 - 4) * (8 + 6)\
* (12 - 5)
```

和

```
a = (3 + 2) * (6 - 4) * (8 + 6) * (12 - 5)
```

是相同的。

下面编写一个最简单的 Python 程序，这里用记事本或其他纯文本编辑器输入下列语句（不能使用 MS Word 等文字处理软件），如图 1.6 所示。



图 1.6 用记事本输入 Python 语句

将上述源代码保存到 D:\pytest 目录下，命名为 hello.py 文件。

（2）运行程序

下面在命令控制台窗口中运行程序。

在命令控制台窗口中，在提示符“D:\pytest>”后输入运行程序命令：

```
python  hello.py
```

注意：如果当前目录不是“D:\pytest”，则应使用 cd 命令，进入到该目录，如图 1.7 所示。

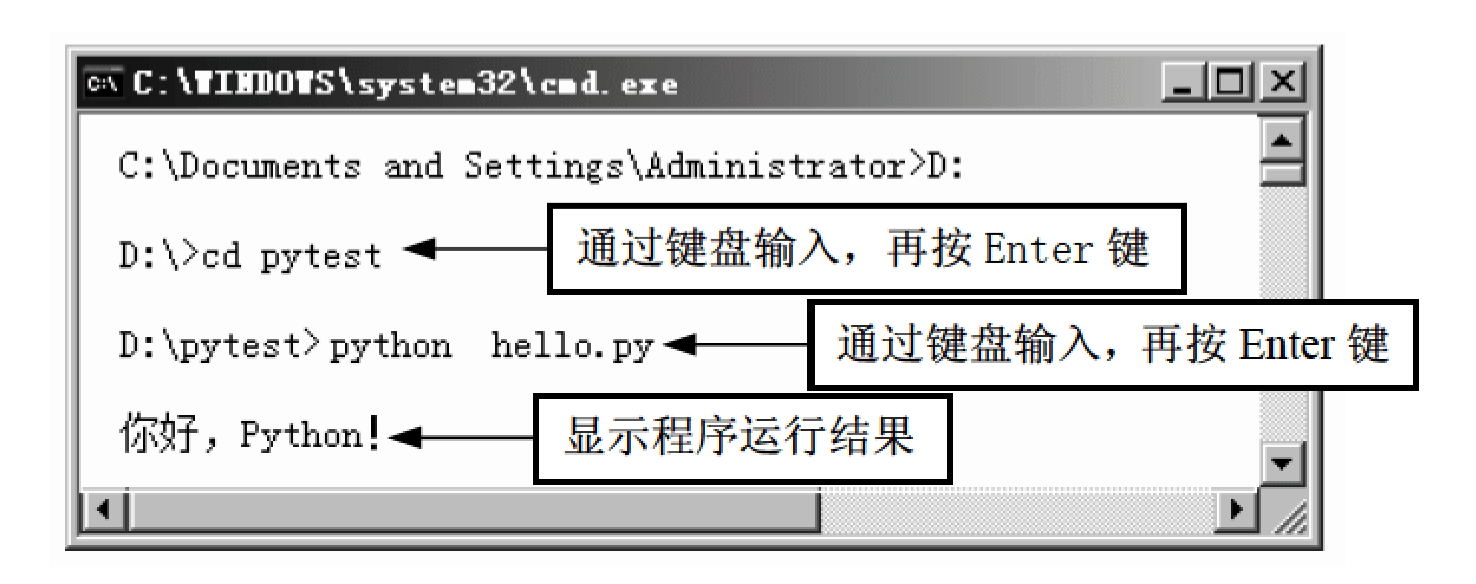


图 1.7 运行 hello.py 程序

1.2.2 Python 编写规范

1. 标识符命名规则

① 文件名、类名、模块名、变量名、函数名等标识符的第一个字符必须是字母表中字母或下画线（_）。

② 标识符的其他部分由字母、数字和下画线组成，且标识符区分大小写字母。

③ 源文件的扩展名为 py。

2. 代码缩进

Python 程序依靠代码块的缩进来体现代码之间的逻辑关系。通常，以 4 个空格或制表符（按 Tab 键)为基本缩进单位。缩进量相同的一组语句，称为一个语句块或程序段。需要注意的是，空格的缩进方式与制表符的缩进方式不能混用。

3. 程序中的注释语句

注释是程序中的说明性文字，是程序的非执行部分。它的作用是为程序添加说明，增加程序的可读性。Python 语言使用两种方式对程序进行注释：

① 单行注释以“#”符号和一个空格开头。如果在语句行内注释（即语句与注释同在一行），注释语句符与语句之间至少要用两个空格分开。

例如：

```
print('Hello')   # 输出显示语句
```

② 多行注释用 3 个单引号'''或 3 个双引号"""将注释括起来。

例如：

```
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
```

4. 代码过长的折行处理

当一行代码较长，需要折行（换行）时，可以使用反斜杠'\'延续行。

例如：

```
io3 = can.create_oval(65,70,185,170, outline='yellow', fill='yellow')
```

可以写成：

```
io3 = can.create_oval(65,70,185,170,  \
                     outline='yellow', \
                     fill='yellow')
```

1.3 编写简单的 Python 程序

【例 1-1】 在命令窗口中显示输出内容的程序。

程序代码如下：

```
str = 'Python 语言入门很简单。\n明白了吗?'
print(str)
```

操作步骤如下：

① 在编辑工具中输入上述程序，如图 1.8 所示。

```
str = 'Python  语言入门很简单。\n 明白了吗?'
print (str)
```

图 1.8 在编辑工具中输入源程序

将输入完成的源程序保存为 ex1_1.py。

② 执行程序：

```
python  ex1_1.py
```

其运行结果如图 1.9 所示。

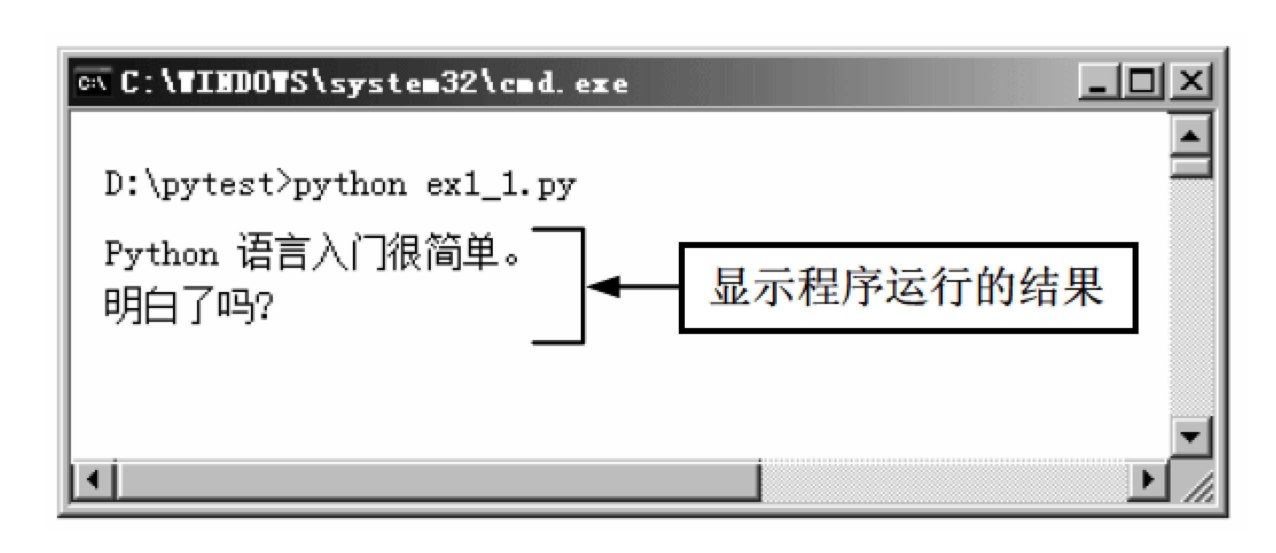


图 1.9　运行结果

【程序说明】

print()为命令窗口输出语句，输出语句中的“\n”是换行符，换行符后面的字符将在下一行显示。

【例 1-2】 输出语句 print()有“原样照印”及简单计算功能。

```
print ('5 + 3 = ', 5+3)
```

用单引号括起来的'5+3='将按原样显示，称为“原样照印”。而没有用单引号括起来的 5+3 将进行加法计算

将其保存为 ex1_2.py。运行程序：

```
python  ex1_2.py
```

其运行结果如图 1.10 所示。

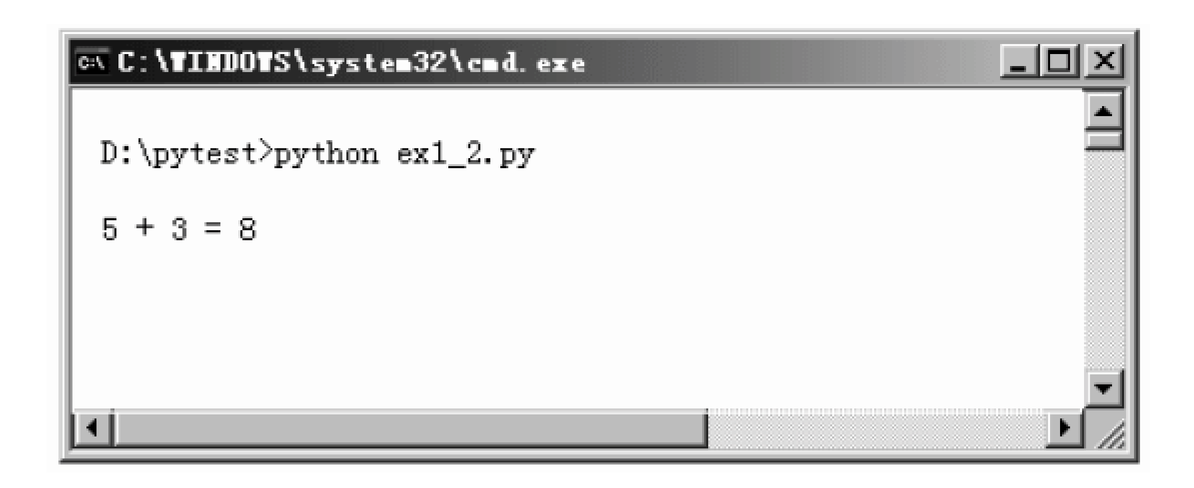


图 1.10　输出语句的“原样照印”及运算功能

【例 1-3】 应用输出语句的“原样照印”功能，输出一个用“*”号组成的三角形。

程序代码如下：

```
print('*')
print('* *')
print('* * *')
print('* * * *')
```

将其保存为 ex1_3.py，运行程序：

```
python  ex1_3.py
```

其运行结果如图 1.11 所示。

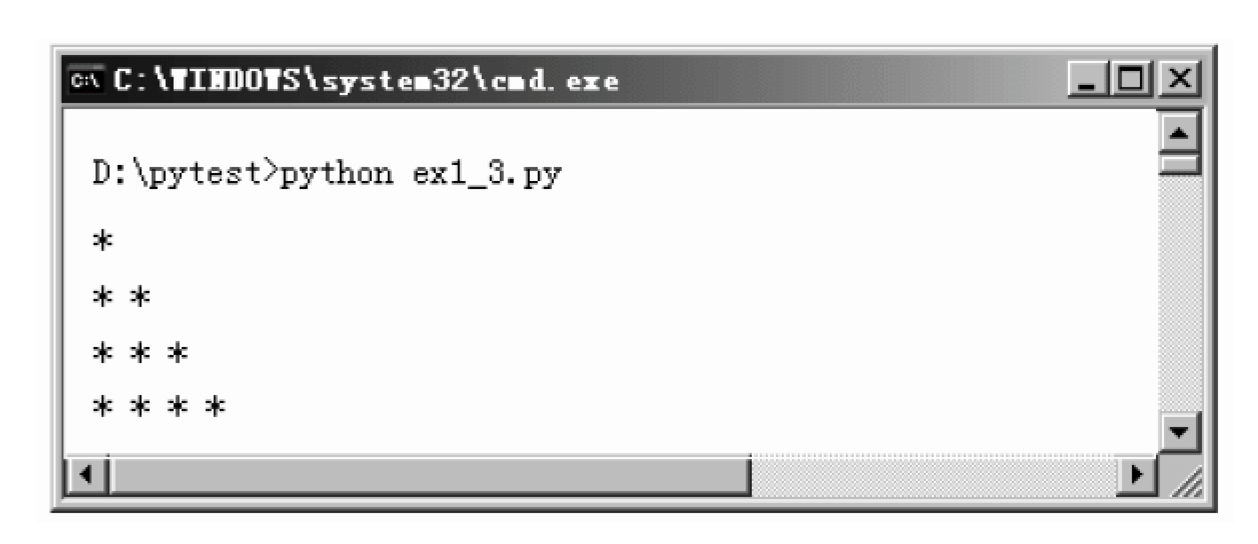


图 1.11　输出用“*”组成的三角形

【例 1-4】 在窗体中显示输出的内容。

程序代码如下：

```
import tkinter
top = tkinter.Tk()
label1 = tkinter.Label(top, text = '在窗体中显示输出内容!')
label1.pack()
top.mainloop()
```

将其保存为 ex1_4.py，运行程序：

```
python  ex1_4.py
```

其运行结果如图 1.12 所示。

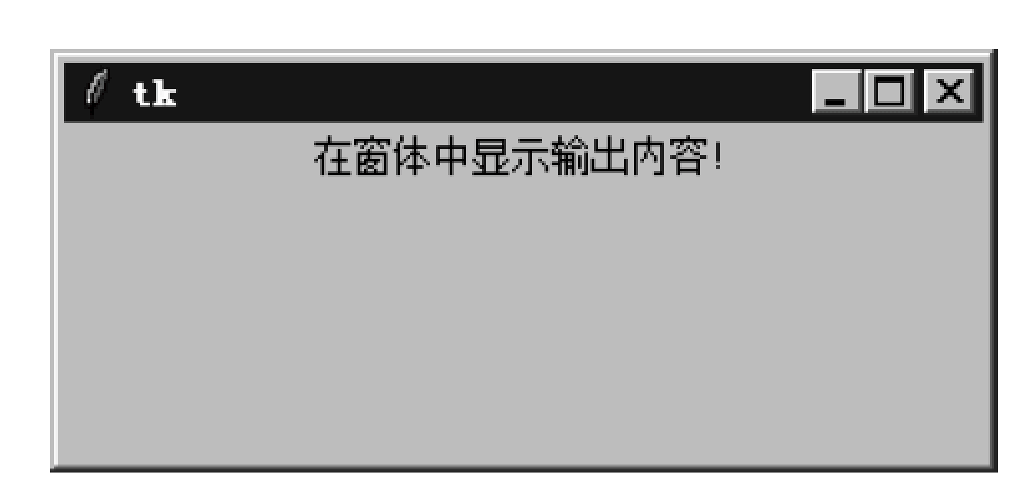


图 1.12　Python 窗体程序的运行结果

【程序说明】

① 程序的第 1 行：

```
import tkinter
```

是一条导入模块的 import 语句。import 语句为编译器找到程序使用的 tkinter 模块。

② 在程序的第 2 行：

```
top = tkinter.Tk()
```

表示创建一个顶层窗体对象。Tk 是模块 tkinter 的类，通过 tkinter.Tk()创建窗体对象。

③ 程序的第 3 行：

```
label1 = tkinter.Label(top, text = '在窗体中显示输出内容!')
```

使用 tkinter 模块的 Label 标签，显示文字内容。

④ 程序的第 4 行：

```
label1.pack()
```

表示把 Label 标签加入到窗体中。pack 是一个顺序排列方式的布局管理器，语句 label1.pack()表示 Label 标签调用 pack()函数将自己加入到窗体容器中。

⑤ 程序的第 5 行：

```
top.mainloop()
```

表示事件循环，使窗体一直保持显示状态。

【例 1-5】 在窗体中显示一幅图像。

程序代码如下：

```
import tkinter
top = tkinter.Tk()
img = tkinter.PhotoImage(file = 'dukou.gif')
label1 = tkinter.Label(image = img, height = 390, width = 330)
label1.pack()
top.mainloop()
```

将其保存为 ex1_5.py，并且在同一文件夹中事先存放了图像文件 dukou.gif。运行程序：

```
python  ex1_5.py
```

其运行结果如图 1.13 所示。



图 1.13　在窗体中显示图像

习　题　1

1．简述 Python 开发环境的建立过程。

2．为什么要为程序添加注释？在 Python 程序中如何为程序添加注释？

3．在计算机中建立一个名为 pytest 的工作目录，在其中保存例 1-1～例 1-5 的源程序，并运行程序。

4．仿照例 1-1，编写并运行 Python 应用程序，显示“多动手练习，才能学好 Python。”。

5．仿照例 1-2，编写并运行 Python 应用程序，计算并显示“1*2*3*4*5”的运算结果。

6．仿照例 1-4，编写并运行 Python 应用程序，在窗体中显示“我对学习 Python 很痴迷!”。

7．仿照例 1-5，编写并运行 Python 应用程序，在窗体中显示一张图片。

第 2 章　Python 语法速览

本章主要介绍 Python 语言中的常量与变量、基本数据类型、运算符、语句、数组等基础知识，熟悉这些知识是正确编写程序的前提条件。程序语言（programming language）本质上就是一种语言，语言的目的在于让人们能与特定对象进行交流，只不过程序语言交流的对象是计算机。学习 Python 语言，就是用 Python 编写程序告诉计算机，希望计算机做哪些事，完成哪些任务。Python 既然是语言，就有其规定的语法规则。本章主要介绍 Python 语言的基本语法和使用规则。

视频录像

2.1　Python 的数据类型

Python 定义了 6 组标准数据类型：

- Number（数字）；
- String（字符串）；
- List（列表）；
- Tuple（元组）；
- Sets（集合）；
- Dictionary（字典）。

1. 数字类型

数字类型包括整数 int、浮点数 float、复数 complex 和布尔值 bool 四种类型。

Python 的数据类型在使用时，不需要先声明，可以直接使用。

例如：

```
x = 13       x为整数
r = 3.14     r为浮点数
a = 3 + 4j   a为复数
```

布尔值类型是一种特殊的数据类型，表示真 True/假 False 值，它们分别映射到整数 1 和 0。

2. 字符串

用单引号或双引号括起来的字符序列称为字符串。

例如，'abc'、'123'、"Hello"和"你好"都是字符串。

在 Python 中定义了很多处理字符串的内置函数和方法（函数是直接调用的，方法需要通过对象用“.”运算符调用），现介绍几个常用的字符串函数和方法。

（1）str()函数

str()函数可以将数字对象、列表对象、元组等转换成字符串。

例如：

```
>>> str(1+2)
 '3'        ← 输出用单引号括起来的字符
>>> str([1,2,3,4])
  '1,2,3,4'
```

（2）find()方法

find()方法可以查找字符子串在原字符串中首次出现的位置，如果没有找到，则返回−1。

例如：

```
>>> s = "ABCDE12345"
>>> s.find("CD")
    2    ← 输出结果，位置从 0 开始起算
```

（3）lower()方法

lower()方法可以将字符串中的大写字母转换为小写字母。

例如：

```
>>> s = "ABCDE12345"
>>> s1 = s.lower()
>>> s1
   abcde12345 ← 输出结果
```

（4）split()方法

split()方法按指定的分隔符将字符串拆分成多个字符子串，返回值为列表。

例如：

```
>>> s = 'AB,CD,123,xyz'
>>> s.split(sep=',')
   ['AB','CD','123','xyz'] ← 输出结果
```

（5）strip()方法

strip()方法用于删除字符串头尾指定的字符（默认为空格）。

例如：

```
>>> str = "*****this is string example...wow!!!*****"
>>> print(str.strip('*'))
this is string example...wow!!! ← 输出结果
```

3. 转义符

在 Python 语言中提供了一些特殊的字符常量，这些特殊字符称为转义符。通过转义符可以在字符串中插入一些无法直接输入的字符，如换行符、引号等。每个转义符都以反斜杠（\）为标志。例如，'\n'代表一个换行符，这里的'n'不再代表字母 n 而作为“换行”符号。

常用的以“\”开头的转义符如表 2.1 所示。

表 2.1　常用转义符

转义符	意义
\b	退格
\f	走纸换页
\n	换行
\r	回车
\t	横向跳格（Ctrl-I）
\'	单引号
\"	双引号
\\	反斜杠

2.2　列表和元组


视频录像

列表是 Python 中使用最频繁的数据类型。系统为列表分配连续的内存空间。

2.2.1　列表定义与列表元素

1. 列表的定义

列表定义的一般形式为：

```
列表名 = [元素0,元素1,…,元素n]
```

说明：

（1）列表名的命名规则跟变量名一样，不能用数字开头。

（2）方括号中的元素之间用逗号分隔。

（3）当列表增加或删除元素时，内存空间自动扩展或收缩。

（4）列表中元素的类型可以不相同，它支持数字、字符串，可以包含列表（称为嵌套列表）。

例如：

```
a1 = [ ];                       # 定义空列表
a2 = [1, 2, 3];                 # 定义三个整数的列表
a3 = ['red', 'green', 'blue'];  # 定义三个字符串的列表
a4 = [5, 'blue', [3, 4]];       # 定义元素类型不相同的嵌套列表
```

2. 列表中元素的访问

（1）列表元素用“列表名[下标]”表示。

例如，有列表

```
a = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

其元素分别为

```
a[0] = 0; a[1] = 1; …; a[9] = 9;
```

（2）用“列表名[起始下标：结束下标 ＋1]”表示列表的片段（列表的部分元素）。例如：设有列表

```
a = [ 0, 1, 2, 3, 'red', 'green', 'blue']
```

用交互方式访问其列表的部分元素。

```
>>> a = [ 0, 1, 2, 3, 'red', 'green', 'blue']
>>> a[0]
0
>>> a[5]
'green'
>>> a[3:]          ← 截取从下标为 3 开始的所有元素
[3, 'red', 'green', 'blue']
>>> a[3:5]         ← 截取从下标为 3 开始到下标为 4 结束的元素
[3, 'red']
>>> a[:2]          ← 截取从首元素开始到下标为 2 结束的元素
[0, 1]
```

2.2.2 列表的操作函数

1. 添加元素

有三个函数可以在列表中添加元素 append()、extend()和 insert()。

（1）用 append()函数在列表末尾添加元素

例如：

```
>>> lst = [ 0, 1, 2, 3]
>>> lst.append(4)      ← 用 append()添加元素
>>> lst
[0, 1, 2, 3, 4]        ← 显示添加后的结果
```

（2）用 extend()函数将另一个列表的元素添加到本列表之后

例如：

```
>>> a = [1, 2, 3]
>>> b = ['x', 'y']
>>> a.extend(b)      ← 用 extend()把列表 b 的元素添加到列表 a 之后
>>> a
[1, 2, 3, 'x', 'y']
```

（3）用 insert()函数将元素插入到列表中指定的某个位置

使用 insert()函数的格式为：

```
insert(下标位置, 插入的元素)
```

例如：

```
>>> lst = [1, 2, 3]
>>> lst.insert(2, 'x')  ← 用 insert()在下标 2 处插入元素'x'
>>> lst
[1, 2, 'x', 3]
```

2. 删除元素

（1）用 del 命令删除列表中指定下标的元素

例如：

```
>>> lst = [1, 2, 3]
>>> del lst[1]  ← 用 del 命令删除下标 1 位置的元素
>>> lst
[1, 3]  ← 显示删除后的结果
```

（2）用 pop()函数删除列表中指定下标的元素

例如：

```
>>> b = ['x', 'y', 'z']
>>> b.pop(1)
'y'  ← 显示被删除的元素
>>> b
['x', 'z']
```

（3）用 remove(x)函数删除列表中所有值为'x'的元素

例如：

```
>>> a = [0, 1, 2, 3]
>>> a.remove(2)  ← 删除列表中值为 2 的元素
>>> a
[0, 1, 3]
```

3. 查找元素位置

用 index()函数可以确定元素在列表中的位置。

例如：

```
>>> str = ['red', 'green', 'blue']
>>> str.index('blue')
2  ← 显示指定元素所在的下标位置
```

4. 对列表元素排序

用 sort()函数可以对列表元素进行排序。sort()函数默认为按升序（从小到大）排序，若要按降序（从大到小）排序，则使用参数 reverse=True。

例如：

```
>>> a = [84, 15, 27, 63, 41]
```

```
>>> a.sort()
>>> a
[15, 27, 41, 63, 84]
>>> a.sort(reverse=True)
>>> a
[84, 63, 41, 27, 15]
```

默认为升序排序

指定为降序排序

5. 清空列表

用 clear()函数可以清空列表中的元素。

例如：

```
>>> a = [0, 1, 2, 3]
>>> a.clear()
>>> a
[]
```

空列表

2.2.3 元组

元组和列表一样，也是一种元素序列。元组是不可变的，元组一旦创建，就不能添加或删除元素，元素的值也不能修改。

1. 元组的创建

用一对括号创建元组。

例如：

```
>>> a = (1, 2, 3)
>>> a
(1, 2, 3)
>>> b = ('数学', '英语', 'C语言')
>>> b
('数学', '英语', 'C语言')
```

创建元组 a

创建元组 b

2. 元组的删除

只能用 del 命令删除整个元组，而不能仅删除元组中的部分元素，因为元组是不可变的。

例如：

```
>>> a = (1, 2, 3)
>>> a
(1, 2, 3)
>>> del a
>>> a
Traceback(most recent call last):
  File "<pyshell#47>", line 1, in <module>
    a
NameError: name 'a' is not defined
```

创建元组 a

删除元组 a

显示元组 a 没有定义的错误提示

2.3　字典和集合



2.3.1　字典

Python 的字典是包含多个元素的一种可变数据类型，其元素由“键：值”对组成，即每个元素包含“键”和“值”两部分。

1. 字典的定义

用大括号{ }把元素括起来就构成了一个 Python 字典对象。

字典中的元素用“字典名[键名]”表示。

例如：

```
>>> D = {'ID': 1001, 'name': '张大山', 'age': 22}    ← 创建字典 D
>>> D
{'ID': 1001, 'name': '张大山', 'age': 22}    ← 显示字典 D 中的元素
>>> D['ID']
1001
>>> D['name']
'张大山'
```
（以上 D['ID'] 与 D['name'] 两例：通过字典元素的键名显示值）

2. 字典元素的修改

通过为键名重新赋值的方式修改字典元素的值。

例如：

```
>>> uil = {'name': 'http', 'port': 80}
>>> uil['name'] = 'ftp'
>>> uil['port'] = 21
>>> uil
{'name': 'ftp', 'port': 21}
```
（uil['name'] = 'ftp' 与 uil['port'] = 21：通过键名为元素重新赋值）

3. 字典元素的添加

添加字典元素，也是使用赋值方式。

例如：

```
>>> D = {'ID': 1001, 'name': '张大山', 'age': 22}
>>> D['sex'] = '男'    ← 新的键名赋值
>>> D
{'ID': 1001, 'name': '张大山', 'age': 22, 'sex': '男'}
```
（'sex': '男'：字典 D 新增的元素）

4. 字典元素的删除

用 del 命令可以删除字典中的元素。

例如：

```
>>> D = {'ID': 1001, 'name': '张大山', 'age': 22}
>>> D.clear()    ← 删除字典 D 中所有的元素
>>> D
```

```
{ }  ← 字典 D 中元素为空
```

2.3.2 集合

集合是一个无序不可重复的序列，是 Python 的一种基本数据类型。

集合分为可变集合（set）和不可变集合（frozenset）两种类型。可变集合的元素是可以添加、删除的，而不可变集合的元素不可添加、不可删除。

1. 集合的定义

集合用一对大括号{ }把元素括起来，元素之间用逗号“，”分隔。

例如：

```
s1 = {1, 2, 3, 4, 5}
s2 = {'a','b','c','d'}
```

上述 s1 和 s2 都是集合。

2. 集合的创建

使用 set()函数创建一个集合。

例如：

```
>>> a = set('abc')
>>> a
{'c','a','b'}  ← 元素无序
```

又如：

```
>>> s = set('book')
>>> s
{'o','k','b'}  ← 元素不重复
```

3. 集合元素的添加

Python 集合有两种方法用于添加元素，分别是 add()和 update()。

（1）使用 add()添加元素

add()把要传入的元素作为一个整体添加到集合中。

例如：

```
>>> a = set('boy')
>>> a.add('python')
>>>a
{'o','y','python','b'}  ← python 作为一个整体添加到集合中
```

（2）使用 update()添加元素

update()把要传入的元素拆分，作为个体添加到集合中。

例如：

```
>>> b = set('boy')
>>> b.update('python')
```

```
>>>b
{'o','y','p','b','t','h','n'}   ← python 拆分成个体添加到集合中
```

4. 集合元素的删除

用 remove()可以删除集合中的元素。

例如：

```
>>> a = set('boy')
>>> a.remove('y')
>>> a
{'o','b'}
```

5. 集合的专用操作符

集合有 4 个专用操作符：&（交集）、|（并集）、–（差集，又称为“相对补集”）和^（对称差分）。

设有两个集合 a 和 b，其关系如下：

- a & b 表示两个集合的共同元素；
- a | b 表示两个集合的所有元素；
- a – b 表示只属于集合 a，不属于集合 b 的元素；
- a ^ b 表示两个集合的非共同元素；

例如：

```
>>> a = set('abc')
>>> b = set('cdef')
>>>
>>> a & b        ← 交集
{'c'}
>>>
>>> a | b        ← 并集
{'c','d','b','f','a','e'}
>>>
>>> a - b        ← 差集
{'a','b'}
>>>
>>> a ^ b        ← 对称差分集
{'d','b','f','a','e'}
```

2.4 程序控制结构


视频录像

2.4.1 顺序控制语句

顺序控制是指计算机在执行这种结构的程序时，从第一条语句开始，按从上到下的顺序依次执行程序中的每一条语句。

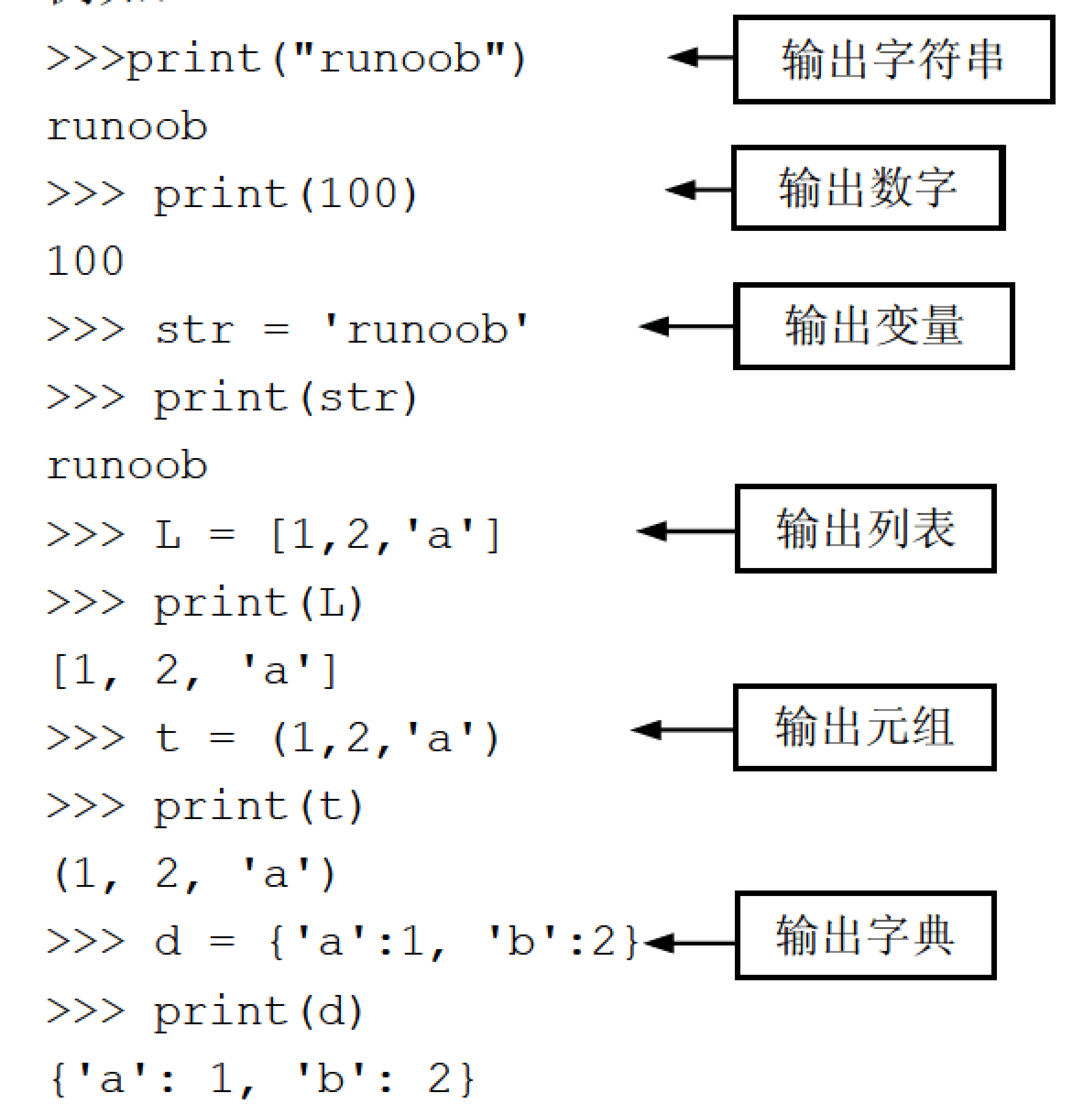
1. 输出语句

在 Python 中使用 print()函数输出数据。

（1）直接输出

字符串、数值、列表、元组、字典等类型都可以用 print()函数直接输出。

例如：



```
>>>print("runoob")
runoob
>>> print(100)
100
>>> str = 'runoob'
>>> print(str)
runoob
>>> L = [1,2,'a']
>>> print(L)
[1, 2, 'a']
>>> t = (1,2,'a')
>>> print(t)
(1, 2, 'a')
>>> d = {'a':1, 'b':2}
>>> print(d)
{'a': 1, 'b': 2}
```


（2）格式化输出

print()函数可以使用 % 格式化输出数据。常用的格式化输出符号如表 2.2 所示。

表 2.2 常用的格式化输出符号

符号	说明
%c	格式化字符及其 ASCII 码
%s	格式化字符串
%d	格式化整数
%e	用科学计数法格式化浮点数

【例 2-1】 格式化输出及控制换行输出示例。

程序代码如下：

```
print('%d %d %s' %(15, 3.14, 12.8))
print('%s %s %s '%('中国', '厦门', '拉萨'))

print('%8.4f'%3.14159)   # 字段宽8，精度4

for i in range(1,4):
    print(i)

for i in range(0,6):
    print(i, end=' ')
```

输出时自动换行

输出时不换行

将程序保存为 ex2_1.py。运行程序：

```
python ex2_1.py
```

程序运行结果如下：

```
15 3 12.8
中国 厦门 拉萨
  3.1416
1
2
3
0 1 2 3 4 5
```

2. 输入语句

在 Python 中，使用 input()函数输入数据。input()函数只能输入字符数据。当需要输入数值型数据时，可以使用 eval()函数将字符转换为数值。

【例 2-2】 从键盘上输入两个数，并计算两数之和。

程序代码如下：

```
print("输入一个整数：")
a = eval(input())
print("输入一个实数：")
b = eval(input())
str = input()
print(str)
c = a + b
print("c =",a,"+",b,"=",c)
```

将程序保存为 ex2_2.py。运行程序：

```
python ex2_2.py
```

程序运行结果如下：

```
输入一个整数：
3 ↙
输入一个实数：
4.5 ↙
运行结果为：↙
运行结果为：
c = 3 + 4.5 = 7.5
```

下画线表示由用户输入的数据，“↙”表示按回车键

【例 2-3】 交换两个变量的值。

在编写程序时，有时需要把两个变量的值互换，Python 在交换值的运算时不需要用中间变量。

程序代码如下：

```
a,b = 3,5
print('a,b =',a,b)
a,b=b,a
print('a,b =',a,b)
```

将程序保存为 ex2_3.py。运行程序：

```
python  ex2_3.py
```

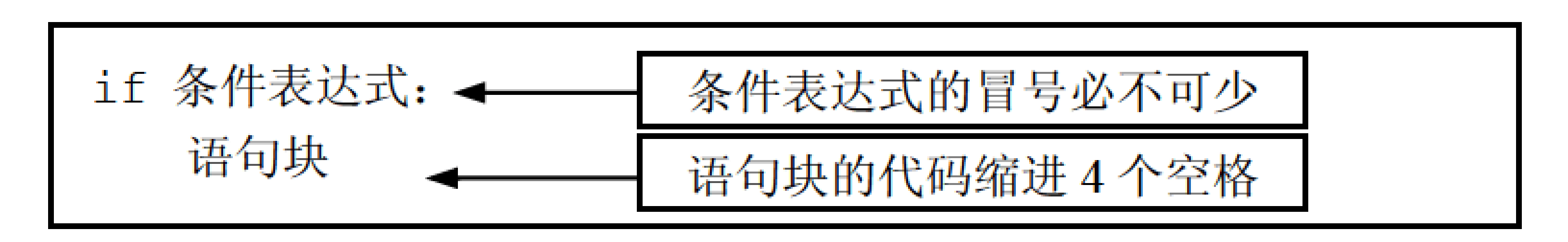
程序运行结果如下：

```
a,b = 3 5
a,b = 5 3
```

2.4.2 if 选择语句

1. 单分支选择结构

if 语句用于实现选择结构。它判断给定的条件是否满足，并根据判断结果决定执行某个分支的程序段。对于单分支选择语句，其的语法格式为：



这个语法的意思是，当条件表达式给定的条件成立时（true），就执行其中的语句块；若条件不成立（false），则跳掉这部分语句，直接执行后续语句，其流程如图 2.1 所示。

【例 2-4】 从键盘任意输入两个整数，按从小到大的顺序依次输出这两个数。

从键盘上输入的两个数 a、b，如果 a < b，本身就是从小到大排列的，可以直接输出。但如果 a > b，则需要交换两个变量的值，其算法流程如图 2.2 所示。

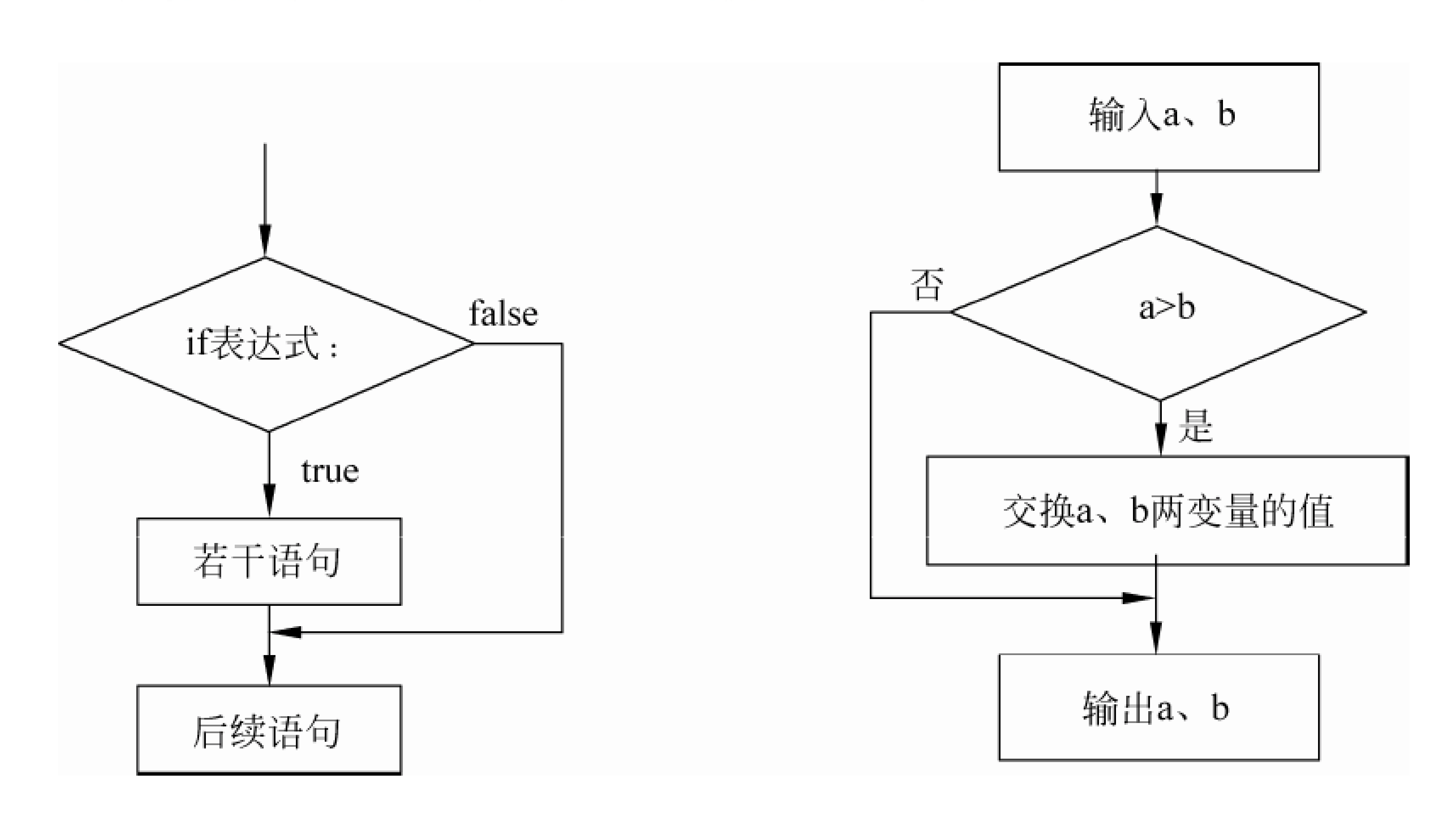


图 2.1 单分支的 if 条件语句　　图 2.2 按从小到大排列的顺序输出两数

程序代码如下：

```
print("输入第一个数：")
a = eval(input())
print("输入第二个数：")
b = eval(input())
print("排序前：", a, b)
if a>b:
   a,b = b,a
print("排序后：", a, b)
```

判断条件，当 a>b 时，执行语句块；当 a<b 时，跳过该语句块

将程序保存为 ex2_4.py。

程序运行结果如下：

```
输入第一个数：8 ↙
输入第二个数：5 ↙
排序前：8, 5
排序后：5, 8
```

【例 2-5】 对给定的三个数，求最大数的平方。

设一变量 max 存放最大数，首先将第一个数 a 放入变量 max 中，再将 max 与其他数逐一比较，较大数则存放到 max 中；当所有数都比较结束之后，max 中存放的一定最大数，其算法流程如图 2.3 所示。

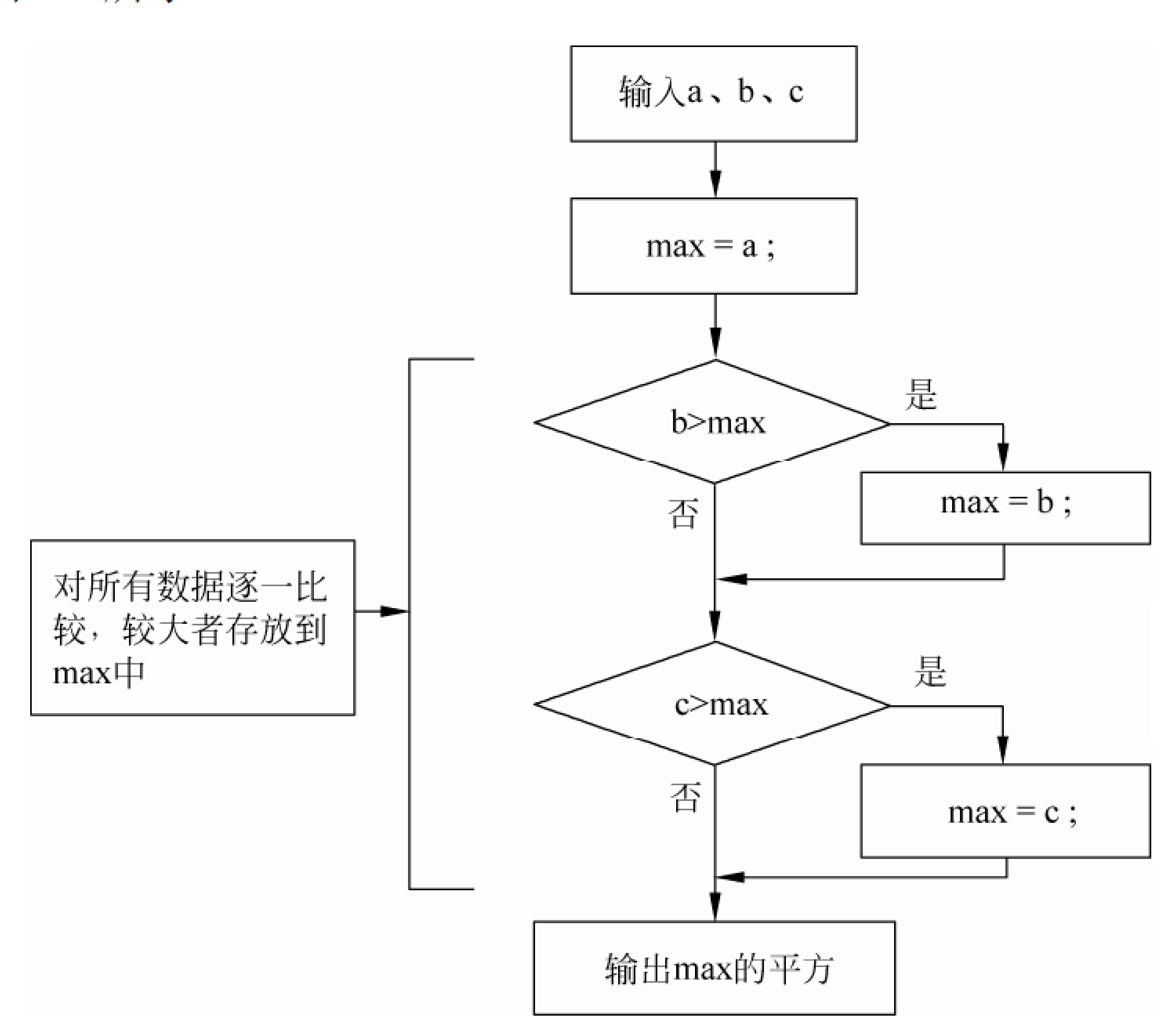


图 2.3　求三个数中最大数的平方

程序代码如下：

```
int a = 5, b = 9, c = 7
max = a;
```

将第一个数 a 赋值给变量 max

```
if b>max:
    max = b
if c>max:
    max = c
print("最大数的平方为：", max * max)
```

第二个数 b 与变量 max 比较，若 b>max，则 b 放到 max 中

第三个数 c 与变量 max 比较，若 c>max，则 c 放到 max 中

将程序保存为 ex2_5.py。

程序运行结果如下：

```
最大数的平方为：81
```

2. 双分支选择结构

有时，需要在条件表达式不成立时执行不同的语句，可以使用另一种双分支选择结构的条件语句，即 if-else 语句。双分支选择结构的语法格式为：

```
if 条件表达式:
    程序段1
else:
    程序段2
```

这个语法的意思是，当条件式成立时(true)，执行语句块 1；否则(else)，就执行语句块 2。对于双分支选择类型的条件语句，其流程如图 2.4 所示。

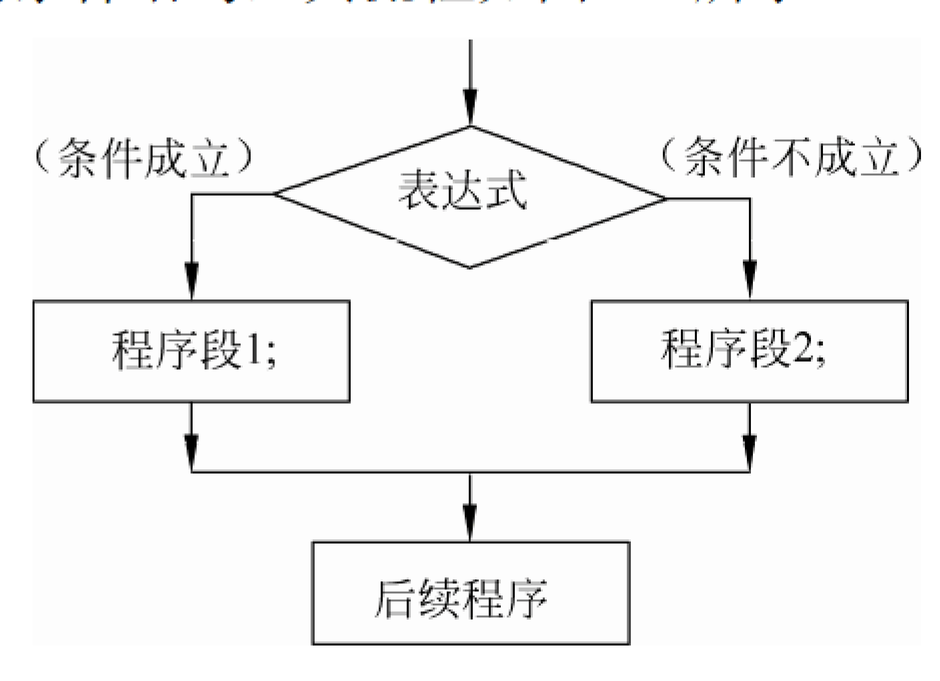


图 2.4　双分支选择结构的条件语句

if-else 语句的扩充格式是 if…else if。一个 if 语句可以有任意个 if…else if 部分，但只能有一个 else。

【例 2-6】 计算 $y=\begin{cases}\sqrt{x^2-25} & x\leqslant -5或x\geqslant 5\\ \sqrt{25-x^2} & -5<x<5\end{cases}$

程序代码如下：

```
import math
x=float(input("请输入x:"))
if x < 5 and x > -5:
    y = math.sqrt(25 - x * x)
else:
    y = math.sqrt(x * x - 25)
```

```
print("y = ", y)
```

将程序保存为 ex2_6.py。

运行程序：

```
python  ex2_6.py
```

程序运行结果如下：

```
4 ↙        ←  从键盘输入 4，并按 Enter 键
y = 3.0
```

3. 多分支选择结构

当处理多种条件问题时，就要使用多分支结构，其语法格式为：

```
if 条件表达式1:
    程序段1
elif条件表达式2:
    程序段2
    ⋮
elif条件表达式n:
    程序段n
else:
    程序段n+1
```

【例 2-7】 将百分制转换为五级记分制。

程序代码如下：

```
a=int(input("请输入百分制成绩："))
b=0
if a>=90:
    b=5
elif a>=80:
    b=4
elif a>=70:
    b=3
elif a>=60:
    b=2
else:
    b=1
print(a,"对应的5分制为：",b)
```

将程序保存为 ex2_7.py。

运行程序：

```
python ex2_7.py
```

程序运行结果如下：

```
请输入百分制成绩：83
83对应的5分制为：4
```

2.4.3 循环语句

在程序设计过程中，经常需要将一些功能按一定的要求重复执行多次，多次重复执行的这一过程称为循环。

循环结构是程序设计中一种很重要的结构。其特点是，在给定条件成立时，反复执行某程序段，直到条件不成立为止。给定的条件称为循环条件，反复执行的程序段称为循环体。

Python 中的循环语句包括 for 语句和 while 语句。

1. for 循环语句

for 循环语句一般形式的语法结构如下：

```
for 循环变量 in range(循环初值, 循环终值, 步长值):
    循环体语句块                              ← 循环体
```

for 语句的执行过程是这样的：首先执行循环变量赋初值，完成必要的初始化工作；再判断循环变量是否小于终值，若循环条件能满足，则进入循环体中执行循环体的语句；执行完循环体之后，循环变量增加一个步长值，以改变循环条件，这一轮循环就结束了。第二轮循环又从判断增加步长值后的循环变量是否小于终值开始，若循环条件仍能满足，则继续循环；否则跳出整个 for 语句，执行后续语句。for 循环语句的执行过程如图 2.5 所示。

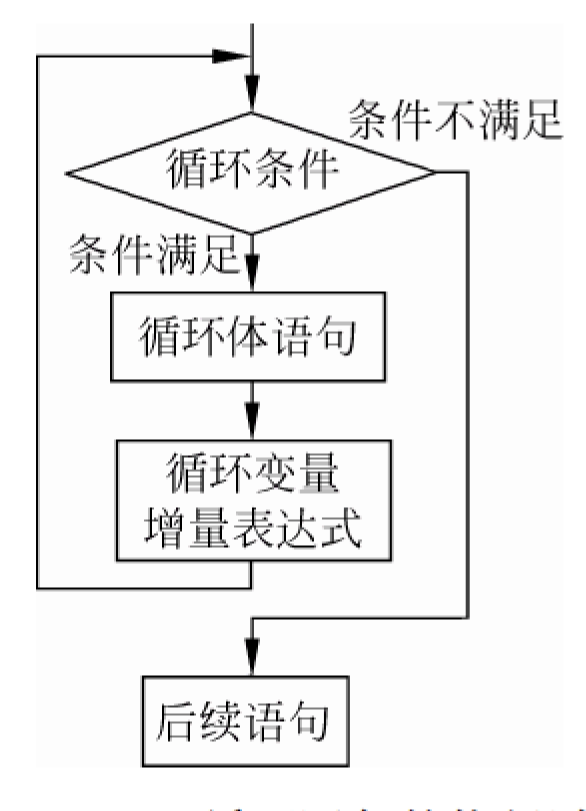


图 2.5　for 循环语句的执行过程

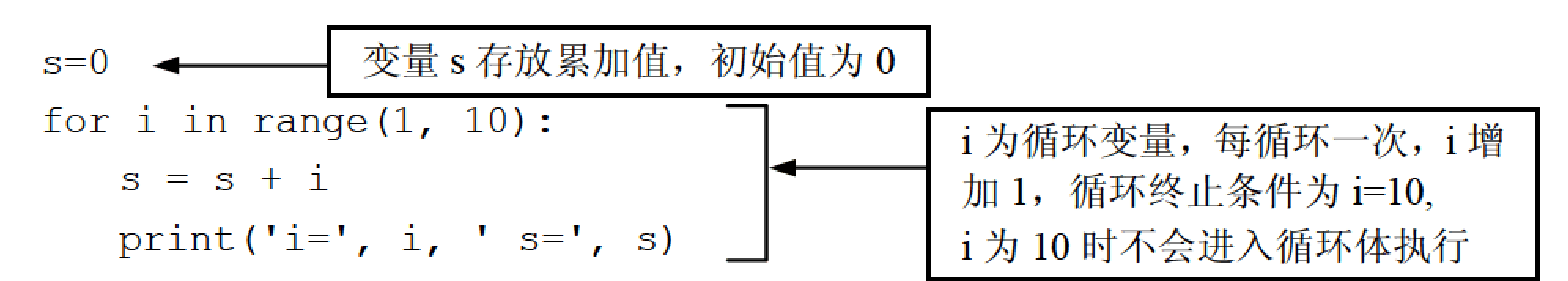
当循环变量的步长值为 1 时，可以省略，即可写成：

```
for 循环变量 in range(循环初值, 循环终值):
    循环体语句块
```

【例 2-8】 求从 1 加到 9 的和。

程序代码如下：

```
i=1
```



```
s=0
for i in range(1, 10):
    s = s + i
    print('i=', i, ' s=', s)
```


将程序保存为 ex2_8.py。

运行程序：

```
python ex2_8.py
```

程序运行结果如下：

```
i=1   s=1
i=2   s=3
i=3   s=6
i=4   s=10
i=5   s=15
i=6   s=21
i=7   s=28
i=8   s=36
i=9   s=45
```

在 for 循环中，循环变量的值在循环体内发生改变，并不会影响循环次数。

【例 2-9】 在循环体内改变循环变量的值，观察循环次数。

程序代码如下：

```
i=1
s=0;
for i in range(1, 10):
    i=i+2
    s = s + i
    print('i=', i, '  s=', s)
```

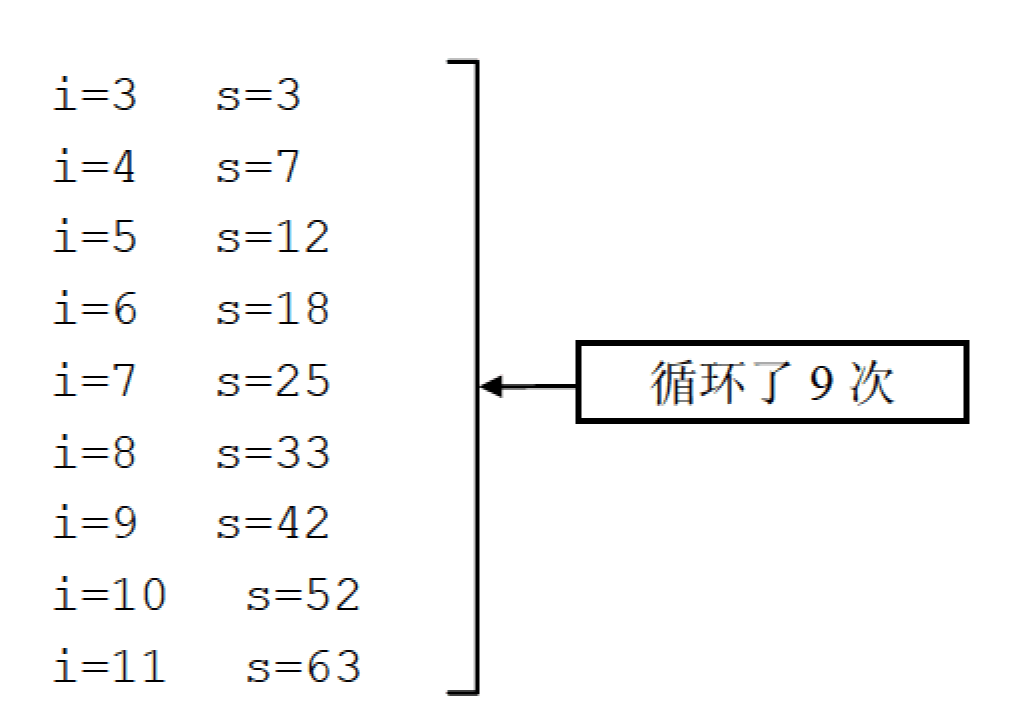
在循环体内改变循环变量的值

将程序保存为 ex2_9.py。

运行程序：

```
python ex2_9.py
```

程序运行结果如下：



```
i=3   s=3
i=4   s=7
i=5   s=12
i=6   s=18
i=7   s=25
i=8   s=33
i=9   s=42
i=10   s=52
i=11   s=63
```

但是，如果把程序改写成：

```
i=1
s=0;
for i in range(1, 10, 2):   ← 通过步长值改变循环变量的值
    s = s + i
    print('i=',i,'  s=',s)
```

则程序运行结果如下：

```
i=1   s=1
i=3   s=4
i=5   s=9      ← 循环了 5 次
i=7   s=16
i=9   s=25
```

在 for 循环中，可以使用 continue 语句结束本次循环，也可以使用 break 语句跳出循环体，从而结束整个循环。

【例 2-10】 计算 10 以内的偶数和。

程序代码如下：

```
i=1
s=0
for i in range(1, 11):
    if i%2 == 1:     ← 循环变量 i 为奇数，则结束
        continue
    s = s + i
    print('i=', i, '  s=', s)
```

在本例中，“i%2 == 1”表示循环变量 i 为奇数。当 i 取奇数时，结束本次循环，继续取下一个数来判断；若 i 为偶数，则计算求和。

将程序保存为 ex2_10.py。

运行程序：

```
python ex2_10.py
```

程序运行结果如下：

```
i=2   s=2
i=4   s=6
i=6   s=12
i=8   s=20
i=10   s=30
```

【例 2-11】 设有列表 s = ['Pytho', 'java', 'C++/C', 'PHP', 'JavaScript']，应用循环遍历列表所有元素，并在屏幕上显示。

程序代码如下：

```
s = ['Pytho', 'Java', 'C++/C', 'PHP', 'JavaScript']
for i in s:        ← 遍历列表所有元素
    print(i)
```

将程序保存为 ex2_11.py。

运行程序：

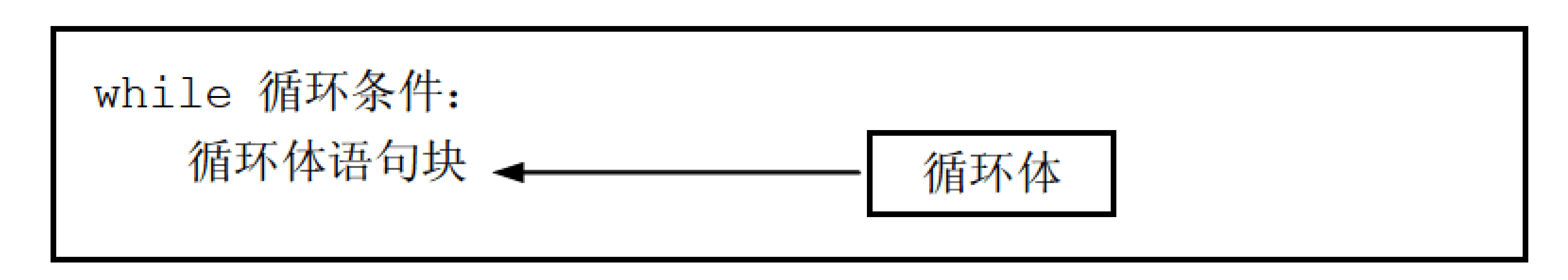
```
python ex2_11.py
```

程序运行结果如下：

```
Pytho
Java
C++/C
PHP
JavaScript
```

2. while 语句

while 循环语句一般形式的语法结构如下：



while 语句的执行过程是这样的：首先判断循环条件是否为 true，若循环条件为 true，则执行循环体中的语句；若为 false，则终止循环。while 循环的流程图如图 2.6 所示。

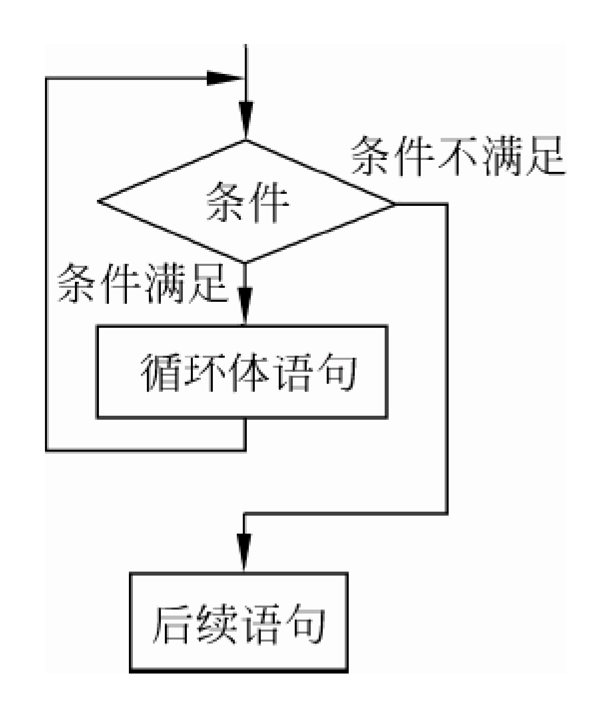


图 2.6　while 循环结构的流程图

【例 2-12】 求 10!。

计算 n!，由于 $p_n = n! = n*(n-1)*(n-2)* \cdots *2*1 = n*(n-1)!$ ，因此可以得到递推公式：

$$p_n = n * p_{n-1}$$
$$p_{n-1} = (n-1) * p_{n-2}$$
$$\vdots$$

$p_1 = 1$

可以用一个变量 p 来存放推算的值，当循环变量 n 从 1 递增到 10 时，用循环执行 p = p *n，每一次 p 的新值都是原 p 值的 n 倍，最后递推求到 10!。

程序代码如下：

```
n=1
p=1
while n<11:
    p = p * n
    print('n=', n, '  p=', p)
        n += 1
```

循环变量 n 在循环体为增加 1

将程序保存为 ex2_12.py。

运行程序：

```
python ex2_12.py
```

程序运行结果如下：

```
n=1   p=1
n=2   p=2
n=3   p=6
n=4   p=24
n=5   p=120
n=6   p=720
n=7   p=5040
n=8   p=40320
n=9   p=362880
n=10   p=3628800
```

3. 循环嵌套

循环可以嵌套，在一个循环体内包含另一个完整的循环，叫作循环嵌套。循环嵌套运行时，外循环每执行一次，内层循环要执行一个周期。

【例 2-13】 应用循环嵌套，编写乘法九九表程序。

算法分析：用双重循环控制输出，用外循环变量 i 控制行数，i 为 1～9。内循环变量 j 控制列数，由于 i＊j＝j＊i，故内循环变量 j 为 1～i。外循环变量 i 每执行一次，内循环变量 j 执行 i 次。

程序代码如下：

```
for i in range(1,10):
 for j in range(1,i+1):
   print(i,'×',j,'=',i*j, end="  ")
 print('')
```

不换行

内循环控制列数

外循环控制行数

其中，print()中的 end=" "表示不换行输出。

将程序保存为 ex2_13.py。

运行程序：

```
python ex2_13.py
```

程序运行结果如下：

```
1×1=1
2×1=2   2×2=4
3×1=3   3×2=6   3×3=9
4×1=4   4×2=8   4×3=12  4×4=16
5×1=5   5×2=10  5×3=15  5×4=20  5×5=25
6×1=6   6×2=12  6×3=18  6×4=24  6×5=30  6×6=36
7×1=7   7×2=14  7×3=21  7×4=28  7×5=35  7×6=42  7×7=49
8×1=8   8×2=16  8×3=24  8×4=32  8×5=40  8×6=48  8×7=56  8×8=64
9×1=9   9×2=18  9×3=27  9×4=36  9×5=45  9×6=54  9×7=63  9×8=72  9×9=81
```

【例 2-14】 应用循环嵌套打印出由“*”组成的直角三角形图形，如图 2.7 所示。

```
*
* *
* * *
* * * *
* * * * *
```

图 2.7　由“*”组成的直角三角形图形

程序代码如下：

```
for i in range(1,6):
    for j in range(1,6):
        if j>i:
            continue
    print('* ' * i, end = " ")
    print('')
```

将程序保存为 ex2_14.py。

运行程序：

```
python ex2_14.py
```

程序运行结果如下：

```
*
* *
* * *
* * * *
* * * * *
```

2.5 函　　数



视频录像

在 Python 中，将用于实现某种特定功能的若干条语句组合在一起，称为函数。本节将简要介绍 Python 中的函数定义及使用方法。

2.5.1 函数的定义与调用

1. 函数定义的一般形式

函数由关键字 def 来定义，其一般形式为：

```
def 函数名(参数列表):
     函数体
    return  (返回值)
```

其中，参数可以为空。当有多个参数时，参数之间用逗号“，”分隔。当函数无返回值，可以省略 return 语句。

【例 2-15】 创建一个名为 Hello 的函数，其作用为输出“欢迎进入 Python 世界”的字符内容。

创建该函数的程序段如下：

```
def Hello():
    print("欢迎进入Python世界")
```

在程序中调用 Hello()函数，将显示“欢迎进入 Python 世界”的字符内容。

【例 2-16】 创建一个名为 sum()的函数，其作用为计算 n 以内的整数之和（包含 n）。

下面为实现计算 n 以内的整数之和的函数程序段：

```
def sum(n):
    s=0
    for i in range(1, n+1):
        s = s + i
    return s
```

2. 函数的调用

在 Python 中，直接使用函数名调用函数。如果定义的函数包含参数，则调用函数时也必须使用参数。

【例 2-17】 创建显示如下排列字符的函数，并编写程序调用该函数。

```
*******************************************
*         欢迎进入学生成绩管理系统          *
*******************************************
```

程序代码如下：

```
def star():
```

```
    str = "*****************************************"
    return str

def  prn():
    print("*          欢迎进入学生成绩管理系统            *")

print(star())
prn()
print(star())
```

将程序保存为 ex2_17.py。
运行程序：

```
python ex2_17.py
```

程序运行结果如下：

```
*****************************************
*          欢迎进入学生成绩管理系统            *
*****************************************
```

【例 2-18】 应用函数，计算 1～100 的和。
程序代码如下：

```
def sum(n):
    s = 0
    for i in range(1, n+1):
        s = s + i
    return s

a=100
ss=sum(a)
print(ss)
```

将程序保存为 ex2_18.py。
运行程序：

```
python ex2_18.py
```

程序运行结果如下：

```
5050
```

2.5.2　局部变量与全局变量

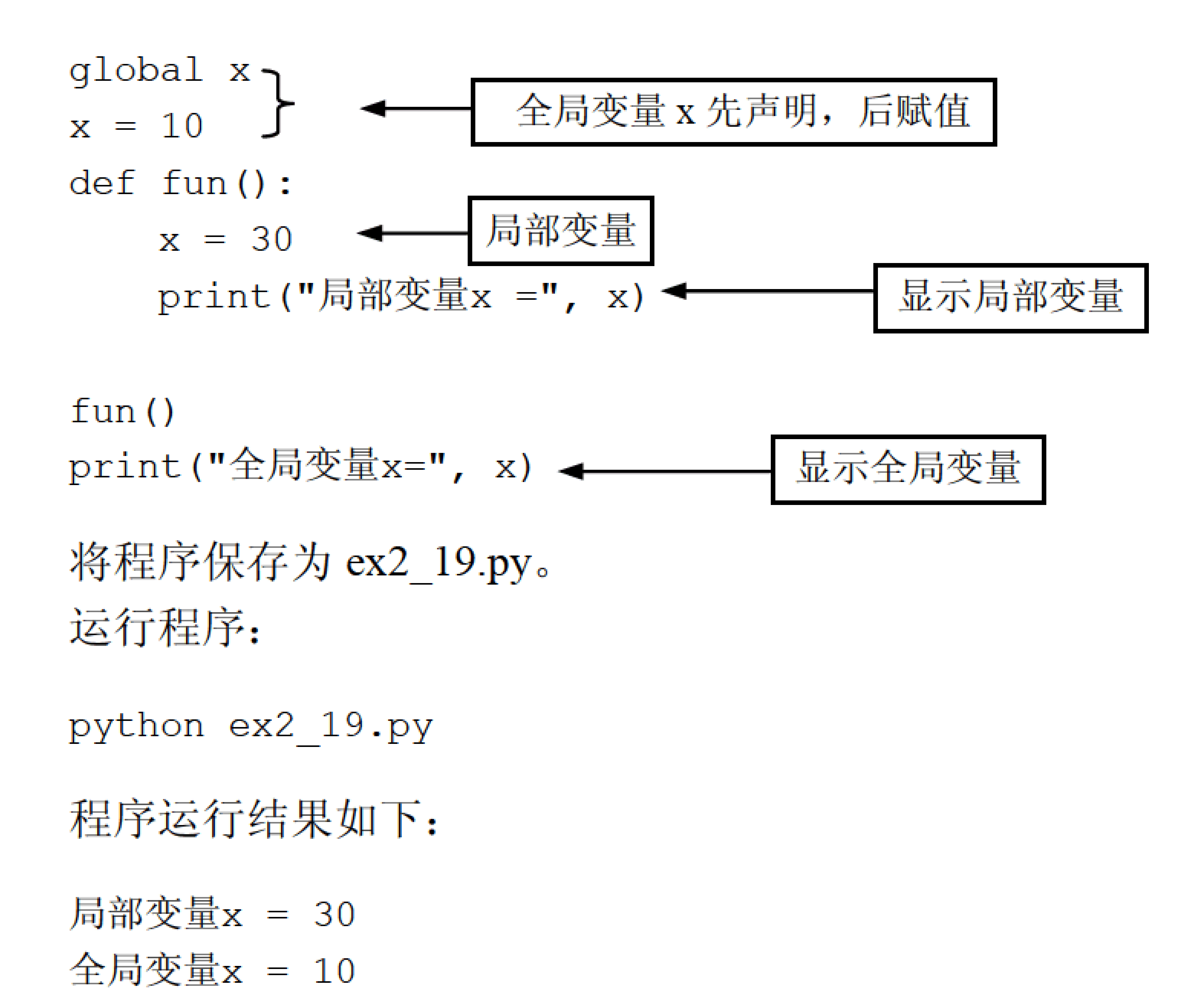
在函数体内部定义的变量或函数参数称为局部变量，该变量只在该函数内部有效。在函数体外部定义的变量称为全局变量，全局变量在变量定义后的代码中都有效。当全局变

量与局部变量同名时，则在定义局部变量的函数中，全局变量被屏蔽，只有局部变量有效。

全局变量在使用前要先用关键字 global 声明。

【例 2-19】 全局变量与局部变量同名的示例。

程序代码如下：



将程序保存为 ex2_19.py。

运行程序：

```
python ex2_19.py
```

程序运行结果如下：

```
局部变量x = 30
全局变量x = 10
```

2.5.3 常用内置函数

Python 内置函数是 python 系统内部创建的，在 Python 的程序中，可以随时调用这些函数，不需要另外定义。

例如，最常见的 print()是内置函数，在程序中直接使用：

```
print("Hello World!")
```

而平方根函数 sqrt()不是内置函数，使用该函数时需要引用 math 模块：

```
import math
y = math.sqrt(25)
```

Python 常用内置函数如表 2.3 所示。

表 2.3 常用内置函数

函数	说明
abs(x)	求绝对值 参数可以是整型，也可以是复数； 若参数是复数，则返回复数的模
divmod(a, b)	返回商和余数的元组
eval(s)	把字符串 s 转换成数值
float([x])	将一个字符串或数转换为浮点数，如果无参数将返回 0.0

续表

函数	说明
int([x[, base]])	将一个字符转换为 int 类型，base 表示进制
long([x[, base]])	将一个字符转换为 long 类型
print()	输出对象，在屏幕上显示
pow(x, y[, z])	返回 x 的 y 次幂
range([start], stop[, step])	产生一个序列，默认从 0 开始
round(x[, n])	四舍五入
sum(iterable[, start])	对集合求和
str(obj)	把数字或其他对象转换成字符串
chr(i)	返回整数 i 对应的 ASCII 字符
bool([x])	将 x 转换为 Boolean 类型
max([整数列表])	返回最大值
min([整数列表])	返回最小值
sum([整数列表])	返回各数之和

【例 2-20】 数学运算函数示例。

程序代码如下：

```
s1=max([1,5,2,9])         # 求最大值
print(s1)

s2=min([9,2,-4,2])        # 求最小值
print(s2)

s3=sum([2,-1,9,12])       # 求和
print(s3)

s4=abs(-5)                # 取绝对值
print(s4)

s5=round(2.6)             # 四舍五入取整
print(s5)

s6=pow(2, 3)              # 计算2的三次方
print(s6)

s7=divmod(9,2)            # 返回除法结果和余数
print(s7)
```

将程序保存为 ex2_20.py。

运行程序：

```
python ex2_20.py
```

程序运行结果如下：

```
9
-4
22
5
3
8
(4, 1)
```

2.5.4 匿名函数 lambda

在 Python 中，可以使用匿名函数。匿名函数即没有函数名的函数。

通常，用 lambda 声明匿名函数。

例如，计算两个数的和，可以写成：

```
add = lambda x, y : x+y
print(add(1,2))
```

输出的结果为 3。

从上面示例可以看到，lambda 表达式的计算结果相当于函数的返回值。

【例 2-21】 用 lambda 表达式，求三个数的和。

程序代码如下：

```
f = lambda x,y,z: x*y*z
print(f(3,4,5))

L = [(lambda x: x**2),
    (lambda x: x**3),
    (lambda x: x**4)]
print(L[0](2),L[1](2),L[2](2) )
```

将程序保存为 ex2_21.py。

运行程序：

```
python ex2_21.py
```

程序的运行结果如下：

```
60
4 8 16
```

2.6 案 例 精 选



视频录像

【例 2-22】 求 50 以内能被 7 整除，但不能同时被 5 整除的所有整数。

程序代码如下：

```
for i in range(1, 51):
    if i%7 == 0 and i%5 != 0:
        print(i)
```

将程序保存为 ex2_22.py。

运行程序：

```
python ex2_22.py
```

程序运行结果如下：

```
7
14
21
28
42
49
```

【例 2-23】 如果一个 3 位数各位数字的立方和等于该数自身，则该数称为“水仙花数”。例如，$153 = 1^3 + 5^3 + 3^3$，所以 153 是一个水仙花数。求 100～1000 所有 “水仙花数”。

程序代码如下：

```
for i in range(100,1000):
    sum = 0                                          # 存放各个位数的立方和
    temp = i
    while temp:
        sum = sum + (temp%10)*(temp%10)*(temp%10)    # 各个数位的立方累加
        temp = int(temp/10)                          # 逐次取数位
    if sum == i:
        print(i)
```

将程序保存为 ex2_23.py。

运行程序：

```
python ex2_23.py
```

程序运行结果如下：

```
153
370
371
407
```

【例 2-24】 设有一份某地连续 10 年 6 月 1 日的气温记录，其数据为（℃）31、30、33、31、28、32、29、33、35、31，试计算其平均气温。

程序代码如下：

```
a = {31,30,33,31,28,32,29,33,35,31}
```

```
s = 0
for i in a:
    s += i
print(int(s/len(a)))
```

将程序保存为 ex2_24.py。

运行程序：

```
python ex2_24.py
```

程序运行结果如下：

```
31
```

【例 2-25】 鸡兔同笼问题。鸡和兔在一个笼子里，从上面数，有 35 个头；从下面数，有 94 只脚。问笼中鸡和兔各有多少只？

设笼中有 x 只鸡，有 y 只兔，则：

```
x + y = 35
2x + 4y = 94
```

程序代码如下：

```
for x in range(1,23):
    y = 35 - x
    if 4*x + 2*y == 94:
        print('兔子有%s只,鸡有%s只'%(x, y))
```

将程序保存为 ex2_25.py。

运行程序：

```
python ex2_25.py
```

程序的运行结果如下：

兔子有 12 只，鸡有 23 只

【例 2-26】 百钱买百鸡问题。公鸡 5 文钱一只，母鸡 3 文钱一只，小鸡 3 只一文钱，用 100 文钱买 100 只鸡，如何买？

设公鸡 x 只，母鸡 y 只，小鸡 z 只，则：

```
x + y + z = 100
5x + 3y + z/3 =100
```

程序代码如下：

```
for x in range(1, 20):       # 从 1 开始买公鸡，不包括20
    for y in range(1, 33):   # 从1开始买母鸡，不包括33
        z = 100 - x - y      # 计算剩余要买多少个小鸡，小鸡的个数要满足3的倍数
        if (z%3 == 0) and (5*x + 3*y + z/3 == 100):# 判断买的计划是否符合条件
```

```
        print('公鸡：%s 母鸡：%s 小鸡：%s'%(x, y, z))
```

将程序保存为 ex2_26.py。

运行程序：

```
python ex2_26.py
```

程序运行结果如下：

```
公鸡：4 母鸡：18 小鸡：78
公鸡：8 母鸡：11 小鸡：81
公鸡：12 母鸡：4 小鸡：84
```

【例 2-27】 老汉卖西瓜，第一天卖西瓜总数的一半多一个，第二天卖剩下的一半多一个，以后每天都是卖前一天剩下的一半多一个，到第 10 天只剩下一个。求西瓜总数是多少？

算法分析：设共有 x 个西瓜，卖一半多一个后，还剩下 x/2 – 1 个，所以，每天的西瓜数可以用迭代表示： $x_n = (x_{n+1} + 1) * 2$。且在卖了 9 天之后（第 10 天），x = 1。这是可以用循环来处理的迭代问题。

程序代码如下：

```
i=1
x=1
while i<=9:
   x=(x+1)*2
   i=i+1
print('西瓜总数：x = ', x)
```

将程序保存为 ex2_12.py。

运行程序：

```
python ex2_12.py
```

程序运行结果如下：

```
西瓜总数：x = 1534
```

【例 2-28】 for 循环语句的应用示例：

（1）使用序列迭代法，显示列表['xyz', 'book', 'hello']。

（2）使用序列索引迭代法，显示列表['c++', 'java', 'python']。

（3）使用数字迭代法，显示 5 个数字。

程序代码如下：

```
# （1）使用序列迭代法
s1 = ['xyz', 'book', 'hello']
for i in s1:
    print(i)
print('\n')
```

```
# （2）使用序列索引迭代法
s2 = ['c++','java', 'python']
for i in range(len(s2)):
    print(i, s2[i])
print('\n')

# （3）使用数字迭代法
x = range(5)
for i in x:
    print(i, x[i])
print('\n')
```

将程序保存为 ex2_28.py。

运行程序：

```
python ex2_28.py
```

程序的运行结果如下：

```
xyz
book
hello

0 c++
1 java
2 python

0 0
1 1
2 2
3 3
4 4
```

【例 2-29】 编写计算 n!的函数。

n!是以递归形式定义的：

$$n! = \begin{cases} 1 & (n = 1) \\ n(n-1) & (n > 1) \end{cases}$$

计算 n!，应先计算(n − 1)!，而计算(n − 1)!，以需要先计算(n − 2)!……依次递推，直到最后变成计算 1!的问题。

根据公式，1! = 1，这是本问题的递归终止条件。由终止条件得到 1!的结果后，再反过来依次计算出 2!，3!，…，n!。

设计算 n!的函数为 fun(n)，当 n>1 时，fun(n) = n * fun(n−1)。即在 fun(n)函数体内将递归调用 fun()自身。

程序代码如下：

```
def fun(n):
    if n == 1:
        return 1
    else:
        return n * fun(n-1)
x = eval(input('请输入n的值：'))
y = fun(x)
print(x,'! = ',y)
```

将程序保存为 ex2_29.py。
运行程序：

```
python ex2_29.py
```

程序运行结果如下：

```
请输入n的值：5  ↙
5! = 120
```

【例 2-30】 编写函数，从键盘输入参数 n，计算斐波那契数列中第一个大于 n 的项。斐波那契数列为 1，1，2，3，5，8，13，…。从第 3 项开始，每一项是前二项之和。
编写程序代码如下：

```
def fun(n):
    a,b = 1,1
    while b<n:
        a,b = b, a+b
    else:
        return b

x = eval(input('请输入n的值：'))
y = fun(x)
print('第一个大于', x, '的项 = ',y)
```

将程序保存为 ex2_30.py。
运行程序：

```
python ex2_30.py
```

程序运行结果如下：

```
请输入n的值：11      ↙
第一个大于 11 的项 =  13
```

【例 2-31】 应用随机函数 random()模拟微信发红包。
使用随机函数 random()需要引用 random 模块。
程序代码如下：

```
import random

def hongbao(total, num):    # 参数total为拟发红包总金额,num为拟发红包数量
    each=[]
    already = 0             # 存放已发红包总金额
    for i in range(1, num):
      # 随机分配红包金额，至少给剩下的人每人留一元钱
        t = random.randint(1, (total - already) - (num - i))
        each.append(t)
        already = already + t
    each.append(total - already) # 所有剩余的钱发给最后一个人
    return each

if __name__ == '__main__':
    total = 50              # 每次发50元
    num = 5                 # 每次发5个红包
    for i in range(10):     # 模拟发10次
        each = hongbao(total, num)
        print(each)
```

将程序保存为 ex2_31.py。

运行程序：

```
python ex2_31.py
```

程序运行结果如下：

```
[27, 4, 10, 5, 4]
[31, 12, 1, 2, 4]
[15, 9, 24, 1, 1]
[9, 23, 1, 9, 8]
[41, 3, 2, 1, 3]
[16, 26, 6, 1, 1]
[14, 20, 7, 6, 3]
[28, 11, 5, 1, 5]
[23, 24, 1, 1, 1]
[44, 2, 2, 1, 1]
```

习 题 2

1．Python 语言中有哪些数据类型？

2．将下列数学表达式写成 Python 中的算术表达式，并编写程序求解。

提示：计算开平方根式，需要导入 math 模块中 sqrt()函数的语句。

```
from math import sqrt
```

（1）$\frac{a+b}{x-y}$　　　　（2）$\sqrt{p(p-a)(p-b)(p-c)}$

3．下面的代码会显示多少次“我对学习 Python 很痴迷”？

```
for i in range(1, 10, 2):
    print("我对学习Python很痴迷")
```

4．下面的代码会显示多少次“我对学习 Python 很痴迷”？

```
for i in 5:
    print("我对学习Python很痴迷")
```

5．编写密码验证程序，用户只有三次输入错误的机会。

6．编写程序，求 $\sum_{k=1}^{10} k^2$ 的值。

7．编写一程序，任意输入三个数，能按大小顺序输出。

8．编写一个 Python 程序，查找 100 以内 3 的倍数的数并将其输出。

9. 编写打印下列图形的程序：

（1）

```
#
##
###
####
```

（2）

```
*******
 *****
  ***
   *
```

第3章 类与模块

3.1 类和对象



视频录像

Python采用了面向对象程序设计的思想，以类和对象为基础，将数据和操作封装成一个类，通过类的对象进行数据操作。

3.1.1 类的格式与创建对象

1. 类的一般形式

类由类声明和类体组成，而类体又由成员变量和成员方法组成，其一般形式如下：

```
class  类名:                ← 类声明
     成员变量               } 类体
     def  成员方法名(self)
```

在类声明中，class是声明类的关键字，表示类声明的开始，类声明后面跟着类名，按习惯类名要用大写字母开头，并且类名不能用阿拉伯数字开头。

在类体中定义的成员方法与在类外定义的函数一般形式是相同的。也就是说，通常把定义在类体中的函数称为方法。

类中的self在调用时代表类的实例，与C++或Java中的this作用类似。

2. 创建对象

类在使用时，必须创建类的对象，再通过类的对象来操作类中的成员变量和成员方法。

创建类对象的格式为：

```
对象名 = 类名()
```

3. 调用成员方法

调用类的方法时，需要通过类对象调用，其调用格式如下：

```
对象名.方法名(self)
```

【例3-1】 编写一个计算两数之和的类。

```
class Myclass:
    def sum(self, x, y):          } 定义类Myclass
        self.x = x
        self.y = y
```

```
        return self.x + self.y

obj = Myclass()
s = obj.sum(3, 5)
print('s =',s)
```

创建类对象，并通过对象调用类的方法

在程序的类定义中，方法 sum(self,x,y)的参数 self 代表类对象自身，self.x = x 即把赋值语句右边的参数 x 值赋值给左边类成员变量 x。为了区分参数及成员变量，在成员变量 x 前面添加 self。

程序的运行结果如下：

```
s = 8
```

4. 类的公有成员和私有成员

在 Python 程序中定义的成员变量和方法默认都是公有成员，类之外的任何代码都可以随意访问这些成员。如果在成员变量和方法名之前加上两个下画线“_ _”作前缀，则该变量或方法就是类的私有成员。私有成员只能在类的内部调用，类外的任何代码都无法访问这些成员。

【例 3-2】 私有成员示例。

```
class testPrivate:
    def _ _init_ _(self, x, y):
        self._ _x = x
        self._ _y = y
    def add(self):
        self._ _s = self._ _x + self._ _y
        return self._ _s
    def printData(self):
        print (self._ _s)

t = testPrivate(3, 5)   # 创建类对象
s = t.add()
t.printData()
print('s = ',s)
```

定义私有方法 _ _init_ _()

定义构造方法，进行初始化

定义类中的普通成员方法

定义类中的普通成员方法

在类的外部调用公有方法

程序的运行结果如下：

```
8
s = 8
```

5. 类的构造方法

在 Python 中，类的构造方法为 _ _init_ _( )，其中方法名开始和结束的下画线是双下画线。构造方法属于对象，每个对象都有自己的构造方法。

如果一个类在程序中没有定义_ _init_ _( )方法，则系统会自动建立一个方法体为空的

_ _init_ _( )方法。

如果一个类的构造方法带有参数，则在创建类对象时需要赋实参给对象。

在程序运行时，构造方法在创建对象时由系统自动调用，不需要用调用方法的语句显式调用。

【例 3-3】 设计一个类 Person。该类有 Name（姓名）和 Age（年龄）两个变量，可以从键盘输入雇员姓名、年龄等信息。

程序代码如下：

```
class Person:
    def _ _init_ _(self, Name, Age):
        self.name = Name
        self.age = Age
    def main(self):
        print(self.name)
        print(self.age)

name = input('input  name:')
age = input('input age:')
p = Person(name, age)
p.main()    # 调用类体中的方法
```

定义构造方法，进行初始化，该构造方法带有参数

定义 main()方法，输出信息

创建对象时，也要带实参。此时系统自动调用构造方法

程序运行结果如下：

```
input name:  sundy
input age:  22
sundy
22
```

6. 析构方法

在 Python 中，析构方法为 _ _del_ _ ( )，其中开始和结束的下画线是双下画线。析构方法用于释放对象所占用的资源，在 Python 系统销毁对象之前自动执行。析构方法属于对象，每个对象都有自己的析构方法。如果类中没有定义_ _del_ _( )方法，则系统会自动提供默认的析构方法。

【例 3-4】 析构方法示例。

程序代码如下：

```
class Mood:
    def _ _init_ _(self,x):
        self.x=x
        print('产生对象',x)

    def _ _del_ _(self):
        print('销毁对象', self.x)

f1 = Mood(1)
```

定义构造方法，创建对象时触发

定义析构方法，释放对象时触发

```
f2 = Mood (2)
del f1
del f2
```

程序运行结果如下：

```
产生对象1
产生对象2
销毁对象1
销毁对象2
```

3.1.2 类的继承

类的继承是为代码复用而设计的，是面向对象程序设计的重要特征之一。当设计一个新类时，如果可以继承一个已有的类，无疑会大幅度减少开发工作量。

在继承关系中，已有的类称为父类或基类，新设计的类称为子类或派生类。派生类可以继承父类的公有成员，但不能继承其私有成员。

在继承中，父类的构造方法__init__()不会自动调用，如果在子类中需要调用父类的方法，可以使用内置函数 super()或通过“父类名.方法名()”的方式实现。

1. 类的单继承

类的单继承的一般形式为：

```
class 子类名(父类名):
    子类的类体语句
```

【例 3-5】 定义一个父类 Person，再定义一个子类 Sunny 继承 Person，并在子类中调用父类的方法。

程序代码如下：

```
class Person:
    def __init__(self, Name, Age):
        self.name = Name
        self.age = Age
    def main(self):
        print('姓名:', self.name)
        print('年龄:', self.age)

class Sunny(Person):
     def __init__(self, name, age, score):
      super(sunny, self).__init__(name, age)
         self.score = score
def prn(self):
         Person.main(self)
         print('成绩:', self.score)

name = input('请输入姓名: ')
```

```
age = input('请输入年龄: ')
score = input('请输入成绩: ')
s = Sunny(name,age, score)       # 实例化子类对象
s.prn()                          # 调用子类的方法
```

程序运行结果如下：

```
请输入姓名: 张大山
请输入年龄: 22
请输入成绩: 88
姓名: 张大山
年龄: 22
成绩: 88
```

2. 类的多继承

Python 支持多继承，多继承的一般形式为：

```
class  子类名(父类名1,父类名2, …, 父类名n):
    子类的类体语句
```

Python 在多继承时，如果这些父类中有相同的方法名，而在子类中使用时没有指定父类名，则 Python 系统将从左往右按顺序搜索。

【例 3-6】 多继承示例。

程序代码如下：

```
class A:
    def __init__(self):
        self.one="第一个父类"                 定义 A 父类
class B:
    def __init__(self):
        self.two="第二个父类"                 定义 B 父类

class C(A,B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)                      定义 A、B 的子类
    def prn(self):
        print(self.one, '\n', self.two)

subc=C()
subc.prn()
```

程序运行结果如下：

```
第一个父类
第二个父类
```

3.1.3 运算符重载

Python 语言提供了运算符重载功能，大大增强了语言的灵活性。运算符重载就是重新定义运算法则。在 Python 中，重载加法运算使用_ _add_ _()方法定义运算法则，重载减法运算使用_ _sub_ _()方法运算法则。

【例 3-7】 设有两个二维元组：（7，10）和（5，−2），它们的加法运算法则为对应元素相加。它们的减法运算法则为对应元素相减。编写程序，计算这两个元组相加、相减的值。

程序代码如下：

```
class Vector:
   def __init__(self, a, b):                    } 定义构造方法
      self.a = a
      self.b = b

   def __str__(self):                                              } 定义元组显示方式
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):                                              } 定义加法运算法则
      return Vector(self.a + other.a, self.b + other.b)

   def __sub__(self,other):                                              } 定义减法运算法则
      return Vector(self.a - other.a, self.b - other.b)

v1 = Vector(7,10)
v2 = Vector(5,-2)
print(v1 + v2)
print(v1 - v2)
```

程序运行结果如下：

```
Vector(12, 8)
Vector(2, 12)
```

3.2 模　　块


视频录像

在 C 语言或 Java 语言中有引用头文件或引用类库中包的方法，从而可以在程序中调用所引用的函数或方法。在 Python 语言中，为了调用系统或其他文件中定义的函数，也有类似的概念，将调用的函数称为模块（module）。

3.2.1 模块的导入

在 Python 中用关键字 import 来导入某个模块，其导入模块的形式有两种。

1. 用 import 形式导入模块

用 import 导入模块的一般形式为：

```
import  模块名
```

例如，要引用模块 math，就可以在文件最开始的地方用

```
import math
```

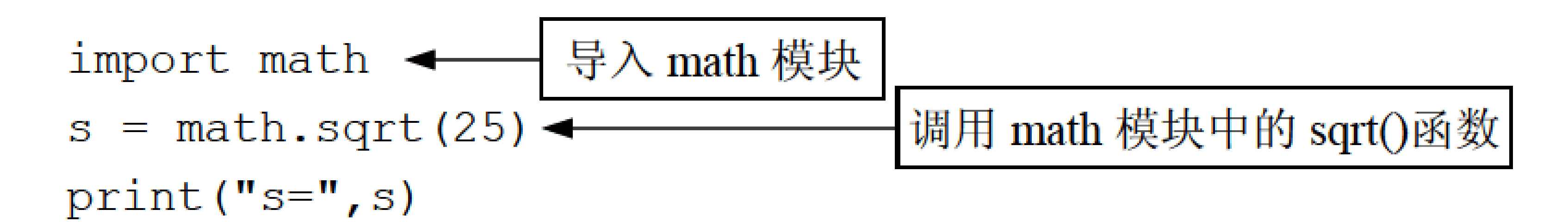
语句来导入。

在调用 import 导入模块的函数时，必须使用以下形式来调用：

```
模块名.函数名
```

例如，调用 math 模块中的 sqrt()函数计算某数的平方根。

程序代码如下：



2. 用 from…import…形式导入模块

用 from…import…导入模块的一般形式为：

```
from  模块名 import  函数名或变量名
```

例如，要引用模块 math 中的 sqrt()函数，可以用

```
from  math  import sqrt()
```

语句来导入。

在调用 from … import …导入模块的函数时，直接使用函数名来调用模块中的函数，而不需要在函数的前面加上模块名。

例如，调用 math 模块中的 sqrt()函数计算某数的平方根。

程序代码如下：

```
from math import sqrt    ← 导入 math 模块中的 sqrt()函数
s = sqrt(25)    ← 调用模块中的函数，前面没有模块名
print("s=",s)
```

3. 导入模块的顺序

当设计的程序需要导入多个模块时，应按照下面的顺序依次导入模块。

（1）导入 Python 系统的标准库模块，如 os、sys 等。

（2）导入第三方扩展库模块，如 pygame、mp3play 等。

（3）导入自己定义和开发的本地模块。

3.2.2 自定义模块

在 Python 中，每个包含函数的 Python 文件都可以作为一个模块使用，其模块名就是文件名。

【例 3-8】 自定义模块使用示例。

① 设有 Python 文件 hello.py，其中包含 hh()函数，代码如下：

```
def hh():
    str="你好, Python! "
    return str
```

② 调用模块 hello 中 hh()函数的程序 ex3_8.py 代码如下：

```
from  Hello import hh  ←  导入 Hello 模块中的 hh()函数

print("引用Hello模块的hh()函数：")
print("______",hh())
```

程序 ex3_8.py 的运行结果如下：

引用 Hello 模块的 hh()函数：

```
______你好, Python!
```

【例 3-9】 编写一个计算两数和的模块，再在另一个程序中调用该模块。

① 编写模块代码，其中包含计算两数之和的函数 sum()，保存为 ex3_9_1.py。

程序代码如下：

```
def sum(n1, n2):
    s = n1 + n2
    return s
```

② 编写调用模块程序 ex3_9_2.py，其程序代码如下：

```
import ex3_9_1
ss = ex3_9_1.sum(3, 5)
print("3 + 5 =",ss)
```

程序 ex3_9_2.py 的运行结果如下：

```
3 + 5 = 8
```

3.2.3 常用标准库模块

Python 系统的标准库中定义了很多的模块，从 Python 语言自身特定的类型和声明，到一些只用于少数程序的模块，总共有 200 多个。

下面介绍一些常用的模块。

1. 核心模块

大部分 Python 程序都有可能直接或间接地使用到这类模块。

（1）os 模块

os 模块中的大部分函数通过对应平台相关模块实现,其常用方法有 open()、file()、listdir()、system()等函数。

（2）sys 模块

sys 模块用于处理 Python 运行时环境。

例如，退出系统时，使用命令：sys.exit(1)。

（3）time 模块

time 模块提供了一些处理日期和时间的函数。例如，用 time()函数获得当前时间。

（4）math 模块

math 模块实现了许多对浮点数的数学运算函数。

例如，使用 math 模块的 sqrt()函数进行开平方根的运算。

程序代码如下：

```
from math import sqrt
x=25
s=sqrt(x)
print("s=",s)
```

2. 线程与进程模块

该部分的模块主要是 Python 系统支持的线程与进程方面的函数。

（1）threading 模块

threading 模块为线程提供了一个高级接口，只需要继承 Thread 类，定义好 run 方法，就可以创建一个新的线程。使用时，首先创建该类的一个或多个实例，然后调用 start 方法。这样，每个实例的 run 方法都会运行在它自己的线程中。具体设计方法在后面的相关章节介绍。

（2）Queue 模块

Queue 模块提供了一个线程安全的队列（queue）实现。通过它可以在多个线程中安全地访问同一个对象。

（3）commands 模块

commands 模块是一些在 UNIX 系统中用于执行外部命令的函数。

3. 网络协议模块

该部分模块主要实现网络访问功能。

（1）socket 模块

socket 模块实现了网络数据传输层的接口，使用该模块可以创建客户端或服务器的套接字 socket 通信。

（2）SocketServer 模块

SocketServer 为各种基于 socket 套接字的服务器提供一个框架，该模块提供大量的类对象，可以用它们来创建不同的服务器。

（3）urllib 模块

urllib 模块为 HTTP、FTP 以及 gopher 提供了一个统一的客户端接口，会自动地根据 URL 选择合适的协议处理器。

（4）cookie 模块

cookie 模块为 HTTP 客户端和服务器提供基本的 cookie 支持。

（5）ftplib 模块

ftplib 模块包含一个文件传输协议（file transfer protocol，FTP）客户端的实现。

（6）httplib 模块

httplib 模块提供一个 HTTP 客户端接口。

（7）webbrowser 模块

webbrowser 模块提供一个到系统标准 Web 浏览器的接口。它提供了一个 open()方法，接受文件名或 URL 作为参数，然后在浏览器中打开它。

4. 正则表达式 re 模块

Python 的 re 模块提供了正则表达式操作所需要的功能接口，利用这个接口，可以直接使用模块中的方法实现对字符串的匹配处理，也可以将匹配模式编译成正则表达式对象。

5. r 原生字符

将在 Python 中有特殊意义的字符（如\b）转换成原生字符（就是去除它在 Python 的特殊意义），不然会与正则表达式有冲突，为了避免这种冲突可以在规则前加原始字符 r。

6. 数据序列化模块 pickle

pickle 模块主要用于序列化对象并保存到磁盘中，并在需要时读取出来，任何对象都可以执行序列化操作。

所谓序列化，是指在保存数据的过程中不丢失其数据类型的过程。列表、字典等数据对象要保存为文件，都必须在进行序列化处理后才能保存到文件中。

主要方法：

（1）pickle.dump(obj, file, [,protocol])

方法的功能：将 obj 对象序列化存入已经打开的 file 中。

参数说明：

obj：想要序列化的 obj 对象。

file：文件名称。

protocol：序列化使用的协议。如果该项省略，则默认为 0。如果为负值或 HIGHEST_PROTOCOL，则使用最高的协议版本。

（2）pickle.load(file)

方法的功能：将 file 中的对象序列化读出。

参数说明：

file：文件名称。

（3）pickle.dumps(obj[, protocol])

函数的功能：将 obj 对象序列化为 string 形式，而不是存入文件中。

参数说明：

obj：想要序列化的 obj 对象。

protocal：如果该项省略，则默认为 0。如果为负值或 HIGHEST_PROTOCOL，则使用最高的协议版本。

（4）pickle.loads(string)

函数的功能：从 string 中读出序列化前的 obj 对象。

参数说明：

string：文件名称。

【例 3-10】 pickle 模块主要函数的应用示例。

程序代码如下：

```
import pickle
dataList = [[1, 1, 'yes'],
            [1, 1, 'yes'],
            [1, 0, 'no'],
            [0, 1, 'no'],
            [0, 1, 'no']]              定义列表
dataDic = { 0: [1, 2, 3, 4],
            1: ('a', 'b'),
            2: {'c':'yes','d':'no'}}        定义字典

# 打开或新建二进制文件（文件操作详见第6章）
fw = open('dataFile.dat','wb')
# 列表序列化并写入文件fw中
pickle.dump(dataList, fw, -1)
# 字典序列化并写入文件fw中
pickle.dump(dataDic, fw)
fw.close()    # 关闭文件

# 打开二进制文件
fr = open('dataFile.dat','rb')
data1 = pickle.load(fr)  ←  读出文件中的第一个序列化对象
print(data1)
data2 = pickle.load(fr)  ←  读出文件中的第二个序列化对象
print(data2)
fr.close()

# 使用dumps()和loads()举例
p = pickle.dumps(dataList) ←  将列表对象序列化为字符串
print(pickle.loads(p))    # 显示转换回的列表对象
p = pickle.dumps(dataDic) ←  将字典对象序列化为字符串
print(pickle.loads(p))    # 显示转换回的字典对象
```

程序运行结果如下：

```
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
{0: [1, 2, 3, 4], 1: ('a', 'b'), 2: {'c': 'yes', 'd': 'no'}}
[[1, 1, 'yes'], [1, 1, 'yes'], [1, 0, 'no'], [0, 1, 'no'], [0, 1, 'no']]
{0: [1, 2, 3, 4], 1: ('a', 'b'), 2: {'c': 'yes', 'd': 'no'}}
```

3.2.4 使用 pip 安装和管理扩展模块

1. 安装 pip

Python 安装第三方的模块，大多使用包管理工具 pip 进行安装。Python 包管理工具 pip 提供了对 Python 包的查找、下载、安装、卸载的功能。

pip 下载地址为 https://pypi.python.org/pypi/pip#downloads。选择 pip-9.0.1.tar.gz 文件进行下载。

下载完成后，将解压的文件存到一个文件夹，使用控制台命令窗口进入解压目录，输入安装命令：

```
python setup.py install
```

pip 安装完后还需要配置环境变量，pip 指令才能生效。找到 python 安装路径下的 scripts 目录下，复制该路径。例如：

```
C:\Users\pandap\AppData\Local\Programs\Python\Python36-32\Scripts
```

将其添加到系统环境变量 path 中。

2. 通过 pip 安装扩展模块

当前，pip 已经成为管理 Python 扩展模块的主要方式。常用 pip 命令如表 3.1 所示。

表 3.1 常用 pip 命令

pip 命令	说明	pip 命令	说明
install	安装模块	list	列出已安装模块
download	下载模块	show	显示模块详细信息.
uninstall	卸载模块	search	搜索模块
freeze	按着一定格式输出已安装模块列表	help	帮助

例如：

① 安装 MySQL 数据库管理模块：

```
pip install pymysql
```

② 安装图形处理库模块：

```
pip install pillow
```

③ 安装 SomePackage 模块：

```
pip install SomePackage
```

④ 卸载 SomePackage 模块：

```
pip uninstall SomePackage
```

⑤ 查看当前已经安装的模块：

```
pip list
```

查看当前已经安装的模块命令运行结果如图 3.1 所示。

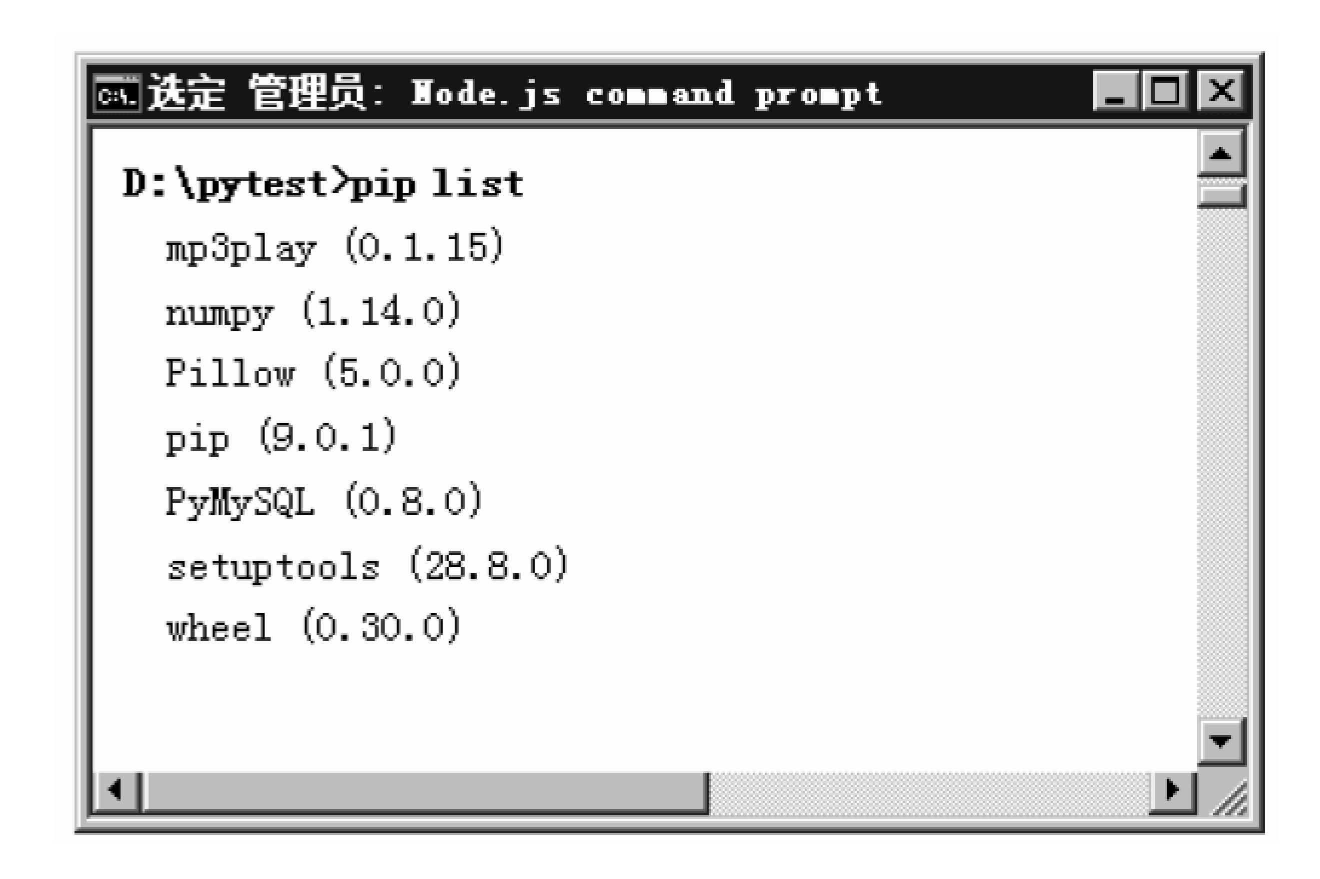


图 3.1　查看已经安装的模块

3.3　案 例 精 选



视频录像

【例 3-11】 设计一个学生类。这个学生类中包含学生的学号、姓名和成绩。计算 3 名学生的成绩平均分。

程序代码如下：

```
class Student:

    def __init__(self, sid, name, scro):
        self.sid = sid
        self.name = name
        self.scro = scro

    def cot(self):
        return self.scro

    def prnid(self):
        print('学号:',self.sid, '姓名:',self.name, '成绩:',self.scro)


stu1 = Student('a1001','张大山',92)
stu2 = Student('a1002','李晓丽',82)
stu3 = Student('a1003','赵志勇',97)

stu1.prnid()
stu2.prnid()
stu3.prnid()
s = stu1.cot() + stu2.cot() + stu3.cot()
print('平均成绩：',int(s/3))
```

程序运行结果如下：

```
学号： a1001 姓名： 张大山 成绩： 92
学号： a1002 姓名： 李晓丽 成绩： 82
学号： a1003 姓名： 赵志勇 成绩： 97
平均成绩： 90
```

【例 3-12】 设计一个学生类。这个学生类中包含学生的学号、姓名和成绩，并能根据学生人数计算成绩平均分。

程序代码如下：

```
class Student:

    def __init__(self):
        self.s = 0              } 在构造方法中初始化变量
        self.count = 0

    def cot(self):
        return self.s/self.count  ◄── 计算平均成绩

    def prnid(self, data):
        for i in data:
          self.sid = i['sid']
          self.name = i['name']     } 获取列表中的字典元素
          self.scro = i['scro']
          self.s = self.s + self.scro
          print('学号:',self.sid, '姓名:',self.name, '成绩:',self.scro)
          self.count += 1

data = [{'sid':'a1001','name':'张大山','scro':92},
       {'sid':'a1002','name':'李晓丽','scro':82},
       {'sid':'a1003','name':'赵志勇','scro':97}]
stu = Student()
stu.prnid(data)
s = stu.cot()
print('平均成绩： ',int(s))
```

程序运行结果如下：

```
学号： a1001 姓名： 张大山 成绩： 92
学号： a1002 姓名： 李晓丽 成绩： 82
学号： a1003 姓名： 赵志勇 成绩： 97
平均成绩： 90
```

习　题　3

1．编写一个具有加、减、乘、除功能的模块，然后通过导入到另一个程序中调用。

2．设计一个商品类，该类有商品编号、品名、价格、数量。应用该类，统计三种商品的总金额。

第 4 章　图形用户界面设计

通过图形用户界面（Graphics User Interface，GUI），用户和程序之间可以方便地进行交互。本章主要介绍如何设计友好的图形用户界面应用程序。

4.1　图形用户界面概述

4.1.1　常用设计图形界面的模块



视频录像

Python 有多种用于设计图形用户界面的模块，常用的模块有如下几种：

- tkinter：使用 Tk 平台，Python 系统自带的标准图形用户界面库。
- wxpython：基于 wxWindows，具有跨平台的特性。
- PythonWin：只能在 Windows 上使用，使用了本机的 Windows GUI 功能。
- JavaSwing：只能用于 Python，使用本机的 Java GUI。
- PyGTK：使用 GTK 平台，在 Linux 上很流行。
- PyQt：使用 Qt 平台，跨平台。

本章主要介绍 tkinter 模块的图形用户界面设计方法。

4.1.2　tkinter 模块

tkinter 模块是 Python 系统自带的标准 GUI 库，具有一套常用的图形组件。tkinter 模块提供的组件如表 4.1 所示。

表 4.1　tkinter 组件

组件	说明
Button	按钮控件，在程序中显示按钮
Canvas	画布控件，显示图形元素，如线条或文本
Checkbutton	多选框控件，用于在程序中提供多项选择框
Entry	输入控件，用于显示简单的文本内容
Frame	框架控件，在屏幕上显示一个矩形区域，多用作容器
Label	标签控件，可以显示文本和位图
Listbox	列表框控件，在 Listbox 窗口小部件是用来显示一个字符串列表给用户
Menubutton	菜单按钮控件，由于显示菜单项
Menu	菜单控件，显示菜单栏，下拉菜单和弹出菜单
Message	消息控件，用来显示多行文本，与 Label 比较类似
Radiobutton	单选按钮控件，显示一个单选的按钮状态
Scale	范围控件，显示一个数值刻度，为输出限定范围的数字区间

续表

组件	说明
Scrollbar	滚动条控件，当内容超过可视化区域时使用，如列表框
Text	文本控件，用于显示多行文本
Toplevel	容器控件，用来提供一个单独的对话框，和 Frame 类似
Spinbox	输入控件，与 Entry 类似，但是可以指定输入范围值
PanedWindow	PanedWindow 是一个窗口布局管理的插件，可以包含一个或多个子控件
LabelFrame	labelframe 是一个简单的容器控件，常用于复杂的窗口布局
tkMessageBox	用于显示应用程序的消息框

使用 tkinter 模块的基本步骤如下：

① 导入 tkinter 模块。

例如：

```
import tkinter
```

或

```
from tkinter import *
```

② 创建一个顶层容器对象。

例如，创建一个窗体对象：

```
win = tkinter.Tk()
```

③ 在顶层容器对象中，添加其他组件。

④ 调用 pack()方法进行容器的区域布局。

⑤ 进入主事件循环。

```
win.mainloop()
```

当容器进入主事件循环状态时，容器内部的其他图形对象则处于循环等待状态，这样才能由某个事件引发容器区域内对象完成某种功能。

4.2 窗体容器和组件

4.2.1 窗体容器和标签组件

1. 窗体

窗体是带有标题、边框的一个顶层容器，在其内部可以添加其他组件，其外观如图 4.1 所示。

设计一个窗体的主要步骤如下：

① 导入 tkinter 包：

```
import tkinter
```

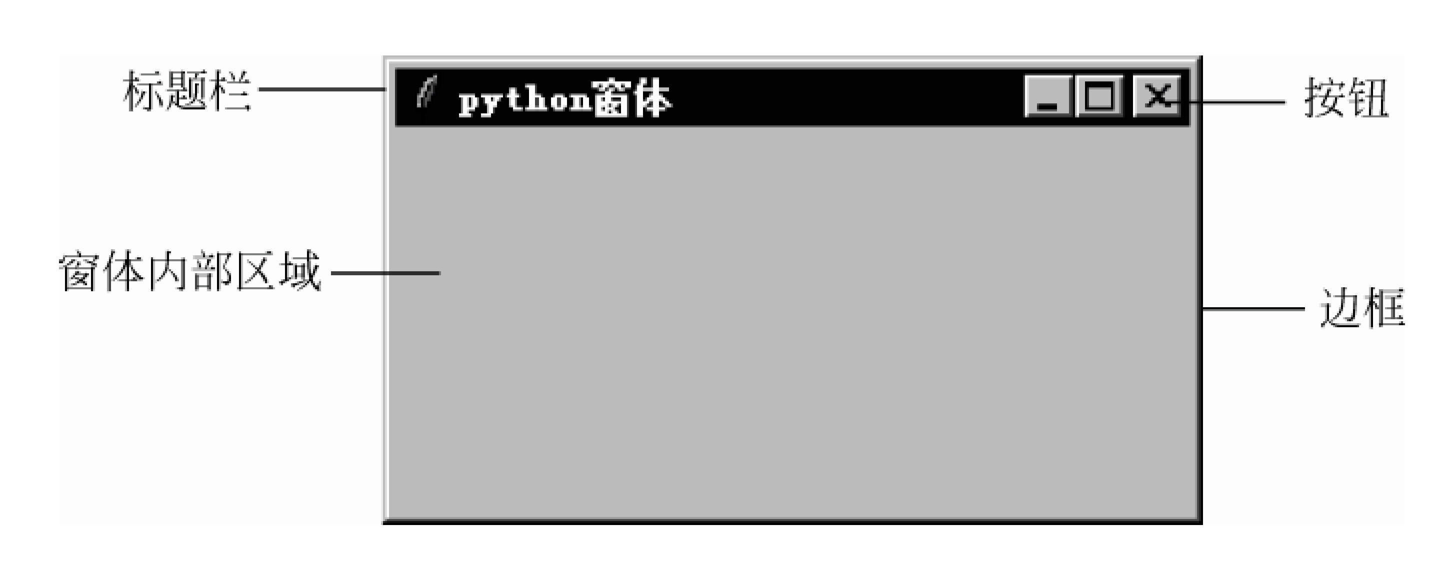


图 4.1　窗体的结构

② 创建窗体对象：

```
win = tkinter.Tk()
```

③ 设置窗体初始的大小（宽×高）和位置（x, y）：

```
win.geometry('宽×高 + x坐标 + y坐标')
```

④ 设置事件循环，使窗体一直保持显示状态：

```
win. mainloop()
```

【例 4-1】 通过 Tk 对象创建一个简单的窗体。

程序代码如下：

```
import tkinter
win = tkinter.Tk()                    # 定义一个窗体
win.title('最简单窗体')               # 定义窗体标题
win.geometry('250×120+50+10')
 # 设置窗体的大小250×120像素和初始位置（50,10）
win.mainloop()                        # 表示事件循环，使窗体一直保持显示状态
```

程序运行结果如图 4.2 所示。



图 4.2　最简单窗体

2. 标签

标签是用于窗体容器中显示文字内容的组件。标签的基本格式为：

```
label = tkinter.Label(容器名称,显示文字或图像内容,显示位置,文字字体、颜色等)
```

标签 Label 的常用属性如表 4.2 所示。

表 4.2　标签 Label 的常用属性

属性选项	说明
text	标签组件的文字内容，可以多行，用'\n'分隔
height	标签组件的文字行数（注意，不是像素）
width	标签组件的文字字符个数（注意，不是像素）
anchor	文本在组件中的位置，默认为“居中”
font	指定文本的字体字号
image	在标签组件中显示图像
textvariable	设置文本变量
bg	设置标签组件的背景颜色
fg	设置标签组件的文字颜色

【例 4-2】 标签应用示例。

程序代码如下：

```
import tkinter
win = tkinter.Tk()                  # 定义一个窗体
win.title('标签示例')               # 定义窗体标题
win.geometry('250×120')             # 定义窗体的大小400×200像素
label = tkinter.Label(win,\
    text ='欢迎进入Python世界!',\
    font='宋体',\                    定义标签 label
    fg='#0000ff'
    )
label.pack()
win.mainloop()
```

程序运行结果如图 4.3 所示。



图 4.3　标签应用示例

4.2.2　按钮和事件处理

1. 按钮对象

当按下应用程序中的按钮时，应用程序能触发某个事件从而执行相应的操作。在 Python 中，tkinter 模块中的 Button 用于构建按钮对象。

（1）按钮 Button 的常用属性

Button 的常用属性如表 4.3 所示。

表 4.3　Button 的常用属性

属性选项	说明
command	单击按钮时，调用事件处理函数
text	设置按钮上的文字
height	设置按钮高度，用文本的字符行数表示
width	设置按钮宽度，用文本的字符个数表示
takefocus	设置焦点
state	设置按钮状态：正常（normal）、激活（active）、禁用（disabled）
bg	设置背景颜色
fg	设置前景颜色

（2）创建按钮对象

创建按钮对象的方法如下：

```
Btn = tkinter.Button(容器, text = "按钮上的文字")
```

由于按钮是一个普通组件，设计时必须放置到一个容器中。下面的示例就是将按钮放置到一个窗体容器内。

【例 4-3】 构造一个带按钮的窗体。

程序代码如下：

```
import tkinter
win = tkinter.Tk()                               # 定义一个窗体
win.title('最简单窗体')                           # 定义窗体标题
win.geometry('400×200')                          # 定义窗体的大小400×200像素
btn = tkinter.Button(win, text = '单击我!')      # 在窗体中添加按钮
btn.pack()
win.mainloop()
```

【程序说明】

由于没有定义按钮的处理事件，因此，单击按钮不会有任何反应。

程序运行结果如图 4.4 所示。



图 4.4　按钮程序运行结果

2. 处理按钮事件

当单击按钮对象 btn 时，按钮的 command 属性触发指定的事件函数，由事件函数处理事件。

【例 4-4】 设计一个按钮事件程序。

程序代码如下：

```
'''
   窗体中的按钮事件示例：
   点击按钮后，弹出一个文本标签
'''
import tkinter
win = tkinter.Tk()                        } 定义窗体
win.title('最简单窗体')
win.geometry('320×180')
t1 = '\n 少壮不努力,老大学程序.'

def mClick():                             } 定义事件函数 mClick
    label1 = tkinter.Label(win, text=t1)
    label1.pack()

Btn = tkinter.Button(win, text = "单击我!", command = mClick)
Btn.pack()                                  ↑ 调用事件函数 mClick

win.mainloop()
```

程序运行结果如图 4.5 所示。

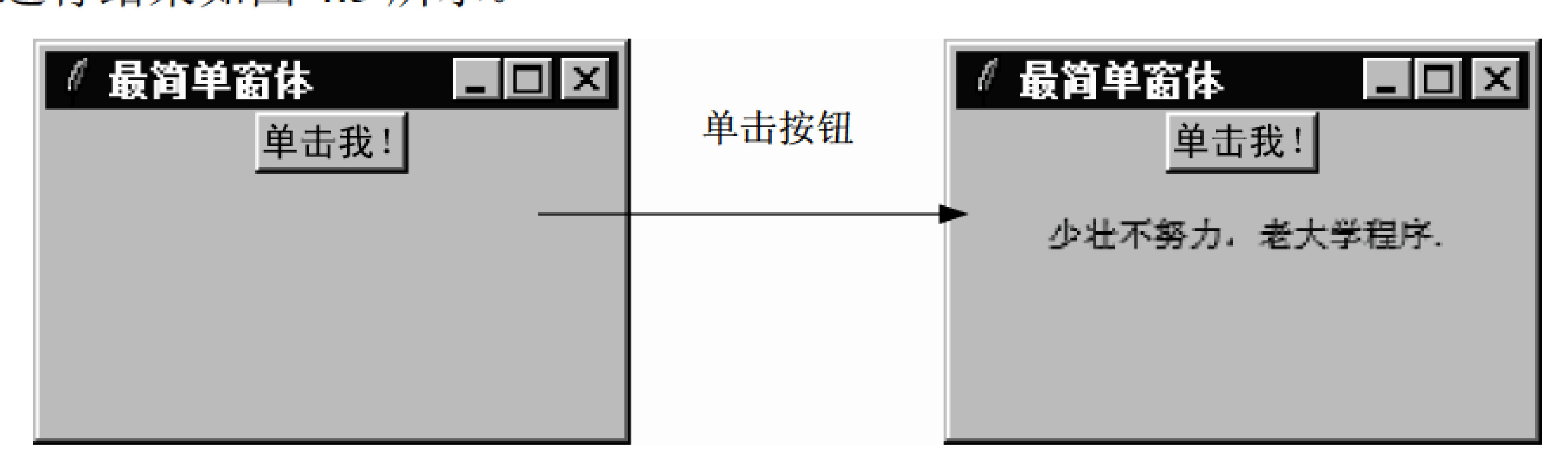


图 4.5　按钮事件

4.3　界面布局管理



视频录像

Python 定义了三种界面布局管理方式，分别是 pack、place 和 grid。

1. pack 布局

pack 布局管理方式按组件的创建顺序在容器区域中排列。

pack 的常用属性有 side 和 fill。

- side 属性：其取值为 top、bottom、left 和 right，分别表示组件排列在上、下、左、右的位置。默认值为 top。
- fill 属性：其取值为 x、y、both，分别表示填充 x（水平）或 y（垂直）方向的空间。

2. place 布局

place 布局管理方式指定组件的坐标位置排列，这种排列方式也称为绝对布局。

3. grid 布局

grid 布局的常用属性有 row（行）、column（列）、rowspan（组件占据行数）、columnspan（组件占据列数）。grid 布局管理方式为网格布局，组件放置在二维表格的单元格中。

【例 4-5】 布局示例。

程序代码如下：

```
from tkinter import Tk,Label

root = Tk()
root.geometry('80×80+10+10') # 80×80为窗体大小，10+10为窗口显示位置
root.title('窗体的布局')

# 填充方式布局
'''
L1=Label(root, text = 'L1', bg = 'red')
L1.pack(fill = 'y')
L2=Label(root, text = 'L2', bg = 'green')
L2.pack(fill = 'both')
L3=Label(root, text = 'L3', bg = 'blue')
L3.pack(fill = 'x')

# 左右方式布局
L1=Label(root, text = 'L1', bg = 'red')
L1.pack(fill = 'y', side = 'left')
L2=Label(root, text = 'L2', bg = 'green')
L2.pack(fill = 'both', side = 'right')
L3=Label(root, text = 'L3', bg = 'blue')
L3.pack(fill = 'x', side = 'left')

# 绝对布局
L4 = Label(root, text = 'L4')
L4.place(x = 3, y = 3, anchor = 'nw')
'''

# Grid 网格布局
L1 = Label(root, text = 'L1', bg = 'red')
L2 = Label(root, text = 'L2', bg = 'blue')
L3 = Label(root, text = 'L3', bg = 'green')
L4 = Label(root, text = 'L4', bg = 'yellow')
L5 = Label(root, text = 'L5', bg = 'purple')

L1.grid(row = 0, column = 0)
L2.grid(row = 1, column = 0)
L3.grid(row = 1, column = 1)
L4.grid(row = 2)
L5.grid(row = 0, column = 3)
```

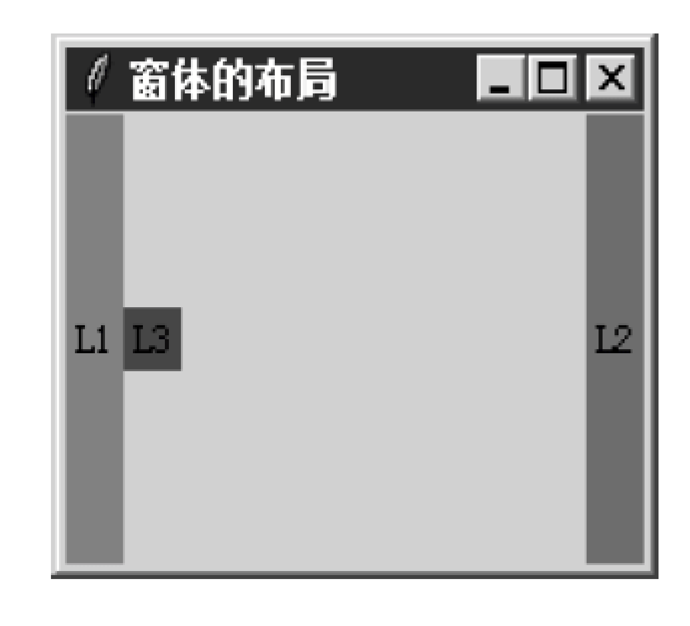
row 为网格的行，column 为网格的列

```
root.mainloop()
```

程序运行结果如图 4.6 所示。




（a）填充方式



（b）左右方式



（c）网格布局

图 4.6　布局排列组件

4.4　文本框组件

1. 文本框的格式

在 Python 中，文本框 Entry 用于接收输入的数据。文本框 Entry 的基本格式为：

```
txt = tkinter.Entry(容器名称,width=宽度, 文字字体、颜色等)
```

文本框 Entry 的常用属性如表 4.4 所示。

表 4.4　文本框 Entry 的常用属性

属性及方法	说明
font	文字字体，值是元组，font = ('字体'，'字号'，'粗细')
foreground	文字颜色，值为颜色或为颜色代码，foreground = 'red'
relief	文本框风格，如凹陷、凸起，取值为 flat/sunken/raised/groove/ridge，　如 relief = 'sunken'
show	指定文本框内容显示为掩码，如密码设为*，show = '*'
state	文本框状态，分为只读和可写，值为 normal/disabled
textvariable	文本框的值，为 StringVar()对象

【例 4-6】 应用布局，设计一个显示学生信息窗体程序（如图 4.7 所示）。



图 4.7　学生信息表

程序代码如下：

```
import tkinter
from tkinter import *

win = tkinter.Tk()     # 定义一个窗体
win.title('学生信息')

vL1 = Label(win, text="学生信息",font = 'Helvetica-36 bold')
L2 = Label(win, text="学号：",font = 'song-20')
L3 = Label(win, text="姓名：",font = 'song-20')
L4 = Label(win, text="专业：",font = 'song-20')
```

设置文本标签

```
L1.grid(row=0,column=1)
L2.grid(row=1)
L3.grid(row=2)
L4.grid(row=3)
```

设置文本标签的排列位置

```
photo = PhotoImage(file='img1.gif')
L_Phot = Label(image=photo)
L_Phot.image = photo
L_Phot.grid(row=0, column=2, columnspan=2, rowspan=4)
```

设置图片及排列位置

```
e1 = Entry(win,width=20, font = 'song -20')
e2 = Entry(win,width=20, font = 'song -20')
e3 = Entry(win,width=20, font = 'song -20')
```

设置文本框

```
e1.grid(row=1, column=1)
e2.grid(row=2, column=1)
e3.grid(row=3, column=1)

win.mainloop()
```

2. 文本框中的内容设置及获取

对文本框 Entry 中文字内容的操作可以使用 StringVar()对象来完成。StringVar()是 tkinter 模块的对象，可以跟踪变量值的变化，把最新的值显示到界面上。把 Entry 的 textvariable 属性设置为 StringVar()，再通过 StringVar()的 get()和 set()函数读取并输出相应内容。这样，文本框中始终显示的值。

【例 4-7】 设计一个密码验证程序。

程序代码如下：

```
from tkinter import *
```

```
win = Tk()
win.geometry('350×116')
win.title('密码验证')

# 提交按钮事件
def mClick():
  txt = txt2.get()
  if(txt == 'abc'):
      txt3.set("欢迎进入本系统")
```

定义按钮事件

```
# 创建几个组件元素
lab1=Label(win, text="请输入用户名: ",font=('华文新魏','16'))
lab2=Label(win, text="请输入密  码: ",font=('华文新魏','16'))
txt1=StringVar()
txt2=StringVar()
txt3=StringVar()
```

声明三个 StringVar()对象，对应 3 个文本组件

```
txt3.set("请输入用户名和密码")
     entry1 = Entry(win,textvariable=txt1, width=16,font=('宋体','16'))
entry2 = Entry(win,textvariable=txt2,width=16,show='*',font=('宋体','16'))
button = Button(win, text='提交', command=mClick,font=('宋体','16'))
lab3=Label(win, textvariable = txt3, relief = 'ridge', width = 30,\
font = ('华文新魏','16'))

# 布局设置
lab1.grid(row=0,column=0)
lab2.grid(row=1,column=0)
entry1.grid(row=0,column=1)
entry2.grid(row=1,column=1)
lab3.grid(row=2,column=0,columnspan=2)
button.grid(row=2,column=2)
```

把区域划分为 3 行 3 列的网格

```
win.mainloop()
```

程序运行结果如图 4.8 所示。

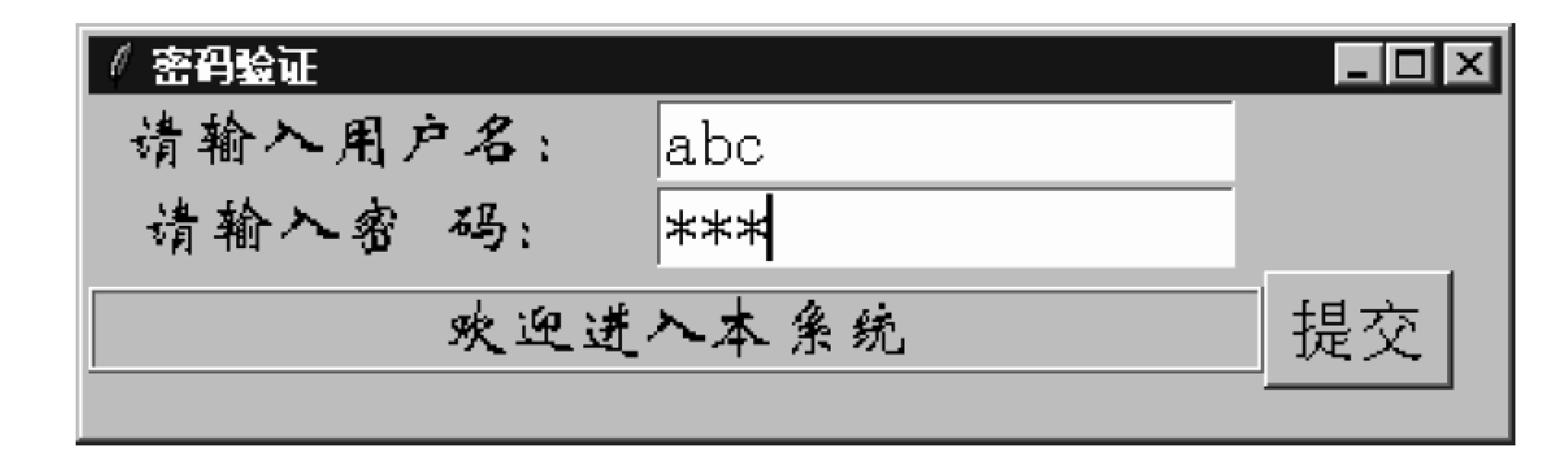


图 4.8 密码框组件程序运行结果

4.5 其他常用组件


视频录像

4.5.1 单选按钮和复选框

单选按钮 Radiobutton 和复选框 Checkbutton 是一组表示多种“选择”的组件。它们都只有两种状态：“选中/未选中”（ON/OFF），其属性和方法都类似，故把它们放在一起介绍。

单选按钮和复选框的常用属性选项如表 4.5 所示。

表 4.5 单选按钮和复选框常用属性选项

属性选项	说明
command	用户选择改变按钮状态，调用相应的方法
text	按钮上的文字内容
variable	组件状态变量，值为 1 时，表示被选中；值为 0 时，表示没选中
onvalue	组件被选中，状态变量值为 1
offvalue	组件没有选中，状态变量值为 0
state	是否可选，当 state='disabled'时，该选项为灰色，不能选择

在创建单选按钮或复选框时，要先声明一个选择状态变量：

```
chVarDis = tk.IntVar()
```

该变量记录单选按钮或复选框是否被选中的状态，可以通过 chVarDis.get()获取其状态，其状态值为 int 类型，勾选为 1；未勾选为 0。

另外，复选框 Checkbutton 对象的 select()方法表示选中，deselect()方法表示未选中。

【例 4-8】 创建包含单选按钮和复选框的窗体。

程序代码如下：

```
from tkinter import *
import tkinter as tk

win = Tk()
win.geometry('400×120')
win.title('单选按钮和复选框示例')

# 显示选择状态的标签
txt = StringVar()
txt.set("请选择")
lab = Label(win,textvariable=txt,relief='ridge',width=30)

# 复选框
chVarDis = tk.IntVar()  # 定义状态变量对象
check1 = tk.Checkbutton(win, text="C语言", variable=chVarDis,\
```

```
state='disabled')
check1.select()    ← 复选框对象的 select()表示为选中

chvarUn = tk.IntVar()  # 定义状态变量对象
check2 = tk.Checkbutton(win, text="Java", variable=chvarUn)
check2.deselect()    ← 复选框对象的 deselect()表示为未选中

chvarEn = tk.IntVar()  # 定义状态变量对象
check3 = tk.Checkbutton(win, text="Python", variable=chvarEn)
check3.select()

# 单选按钮
chk = ["鲜花", "鼓掌", "鸡蛋"]# 定义几个选项的全局变量

# 单选按钮回调函数,当单选按钮被单击执行该函数
def radCall():
    radSel = radVar.get()
    if radSel == 0:
        txt.set(chk[0])
    elif radSel == 1:
        txt.set(chk[1])
    elif radSel == 2:
        txt.set(chk[2])
    print(radVar.get())

radVar = tk.IntVar()  # 定义状态变量对象
for i in range(3):
   curRad = tk.Radiobutton(win, text = chk[i],\
                     variable = radVar, value = i,\
                     command = radCall)
   curRad.grid(column = i, row=5, sticky = tk.W)
```

用循环控制单选按钮组，当其中某个单选按钮被选中时，触发属性 command 对应的方法

```
# 布局设置
lab.grid(row=0,column=0,columnspan=3)
check1.grid(column=0, row=4, sticky=tk.W)
check2.grid(column=1, row=4, sticky=tk.W)
check3.grid(column=2, row=4, sticky=tk.W)
```

其中，sticky=tk.W 为设置行列对齐方式，N 表示北/上对齐，S 表示南/下对齐，W 表示西/左对齐，E 表示东/右对齐

```
win.mainloop()
```

程序运行结果如图 4.9 所示。

4.5.2 标签框架、下拉列表框和滚动文本框

1. 标签框架 LabelFrame

标签框架 LabelFrame 是一个带边框的容器，可以在该容器中放置其他组件。



图 4.9　单选按钮和复选框示例

标签框架 LabelFrame 的构造方法为：

```
ttk.LabelFrame(上一级容器, text="标签显示的文字内容")
```

2.　下拉列表框 Combobox

下拉列表框 Combobox 是常用的一种选值组件，使用下拉列表框时要先声明一个取值变量：

```
number = tk.StringVar()
```

该变量记录在下拉列表框预设的值中所选取的字符值，在下拉列表框中预设的值为一个元组。

下拉列表框 Combobox 的构造方法为：

```
ttk.Combobox(容器, width=宽度, textvariable=取值变量)
```

3.　滚动文本框 scrolledtext

滚动文本框 scrolledtext 是一个带滚动条的文本框，可以输入多行文本内容。其构造方法为：

```
scr = scrolledtext.ScrolledText(容器, width=文本框宽度, height=文本框高度)
```

【例 4-9】　标签框架、下拉列表框和滚动文本框示例。

程序代码如下：

```
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext                        # 导入滚动文本框的模块

win = tk.Tk()                                           # 创建一个窗体对象
win.title("Python 组件演示")                             # 设置窗体标题

# 创建一个标签框架容器,
monty = ttk.LabelFrame(win, text=" 标签框架 ")           # 创建一个容器,其父容器为win
monty.grid(column=0, row=0, padx=10, pady=10)
                                    # padx和pady为容器外围需要留出的空余空间
aLabel = ttk.Label(monty, text="A Label")
```

```
ttk.Label(monty, text="请选择一个数字: ").grid(column=1, row=0)

# 按钮的方法
def clickMe():
  action.configure(text='Hello'+' '+numberChosen.get())  # 设置按钮上的文字

# 按钮
action = ttk.Button(monty, text="单击我!", command=clickMe)
action.grid(column=2, row=1)

# 创建一个下拉列表框
num = tk.StringVar()
numberChosen=ttk.Combobox(monty,width=12,textvariable=num,state='readonly')
numberChosen['values'] = (1, 2, 4, 42, 100)◄── 设置下拉列表框的值
numberChosen.grid(column=1, row=1)
numberChosen.current(0)                    # 设置下拉列表默认值

# 滚动文本框
scrolW = 30                                # 设置文本框的长度
scrolH = 3                                 # 设置文本框的高度
scr = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH)
scr.grid(column=0, columnspan=3)           # columnspan 将3列合并成一列

win.mainloop()                             # 当调用mainloop()时,窗口才会显示出来
```

程序运行结果如图 4.10 所示。



图 4.10　标签框架、下拉列表和滚动文本框示例

4.6　菜单与对话框



视频录像

4.6.1　菜单

一个窗体的菜单由菜单条、菜单和菜单项组成，窗体中放置菜单条，菜单条中放置菜单，菜单中放置菜单项，而菜单项引发相应的动作事件，如图 4.11 所示。



图 4.11　菜单由菜单条、菜单、菜单项组成

创建菜单的主要步骤如下：

（1）创建菜单条对象

```
menubar=Menu(窗体容器)
```

（2）把菜单条放置到窗体中

```
窗体容器.config(menu=menubar)
```

（3）在菜单条中创建菜单

```
菜单名称=Menu(menubar, tearoff=0)
```

其中，tearoff 取值 0 表示菜单不能独立使用。

（4）为菜单添加文字标签

```
menubar.add_cascade(label="文字标签", menu=菜单名称)
```

（5）在菜单中添加菜单项

```
菜单名称.add_command(label="菜单项名称", command=功能函数名)
```

【例 4-10】 菜单应用示例。

程序代码如下：

```
from tkinter import *

class MenuDemo:
    def hello(self):
        print("hello!")

    def __init__(self):
        window = Tk()
        window.title("菜单演示")
        menubar = Menu(window)                  # 定义菜单条
        window.config(menu = menubar)

        # 创建下拉菜单,并添加到菜单条
        # 在菜单条中定义"操作"菜单
        operationMenu = Menu(menubar, tearoff = 0)
        menubar.add_cascade(label = "操作", menu = operationMenu)
```

```
        # 在菜单添加菜单项
        operationMenu.add_command(label = "加", command = self.add)
        operationMenu.add_command(label="减", command=self.subtract)
        operationMenu.add_separator()            # 菜单项的分隔符
        operationMenu.add_command(label = "乘", command = self.multiply)
        operationMenu.add_command(label="除", command=self.divide)

        exitMenu = Menu(menubar, tearoff = 0)   # 在菜单条中定义"退出"菜单
        menubar.add_cascade(label = "退出", menu = exitMenu)
        exitMenu.add_command(label = "退出", command = window.quit)

        mainloop()
    def add(self):
        print("相加")
    def subtract(self):
        print("相减")
    def multiply(self):
        print("相乘")
    def divide(self):
        print("相除")

MenuDemo()
```

“关闭”窗体，退出系统

程序运行结果如图 4.12 所示。

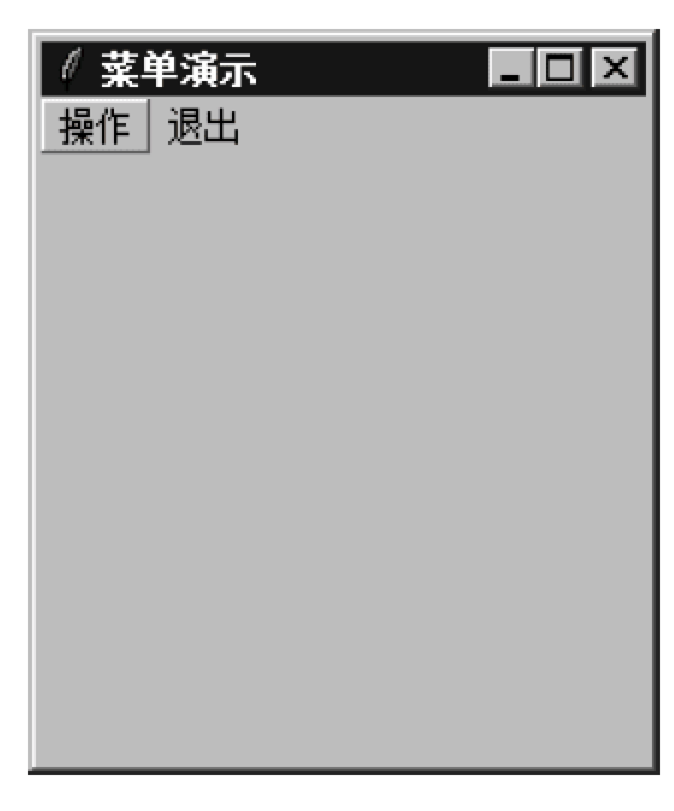


图 4.12　菜单演示示例

4.6.2　对话框

tkinter 提供了三种标准的对话框模块：

- 消息对话框 messagebox；
- 文件对话框 filedialog；
- 颜色选择对话框 colorchooser。

1.　无返回值的消息对话框

消息对话框分为无返回值的对话框和有返回值的对话框，这两种消息对话框的导入模

块语句都是一样的。

（1）消息对话框的导入模块语句

```
import tkinter
import tkinter.messagebox #这是消息框，对话框的关键
```

（2）消息提示框

```
tkinter.messagebox.showinfo('提示','人生苦短')
```

消息提示框如图 4.13 所示。

（3）消息警告框

```
tkinter.messagebox.showwarning('警告','明日有大雨')
```

消息警告框如图 4.14 所示。

（4）错误消息框

```
tkinter.messagebox.showerror('错误','出错了')
```

错误消息框如图 4.15 所示。



图 4.13 消息提示框



图 4.14 消息警告框



图 4.15 错误消息框

【例 4-11】 无返回值消息对话框示例。

程序代码如下：

```
import tkinter
import tkinter.messagebox

def but_info():
    tkinter.messagebox.showinfo('提示', '人生苦短')

def but_warning():
    tkinter.messagebox.showwarning('警告', '明日有大雨')

def but_error():
    tkinter.messagebox.showerror('错误', '出错了')
```

```
root=tkinter.Tk()
root.title('消息对话框')                        # 标题
root.geometry('400×400')                        # 窗体大小
root.resizable(False, False)                    # 固定窗体
tkinter.Button(root, text='消息提示框',command=but_info).pack()
tkinter.Button(root, text='消息警告框',command=but_warning).pack()
tkinter.Button(root, text='错误消息框',command=but_error).pack()
root.mainloop()
```

运行程序，在窗体中，单击“消息提示框”按钮，则弹出消息提示框，见图 4.13。单击“消息警告框”按钮，则弹出消息警告框，见图 4.14。单击“错误消息框”按钮，则弹出错误消息框，见图 4.15。

2. 有返回值的消息对话框

（1）askokcancel()

askokcancel()函数在对话框中显示“确定”和“取消”按钮，其返回值分别为 True 或 False，如图 4.16 所示。

例如：

```
a = tkinter.messagebox.askokcancel('提示', '要执行此操作吗')
print(a)
```

当单击对话框的“确定”按钮时，程序结果为 True；当单击对话框的“取消”按钮时，程序结果为 False。

（2）askquestion()

askquestion()函数在对话框中显示“是”和“否”按钮，其返回值分别为 yes 或 no，如图 4.17 所示。



图 4.16　消息框中显示“确定”和“取消”按钮

图 4.17　消息框中显示“是”和“否”按钮

例如：

```
a = tkinter.messagebox.askquestion('提示', '要执行此操作吗')
print(a)
```

当单击对话框的“是”按钮时，程序结果为 yes；当单击对话框的“否”按钮时，程序结果为 no。

（3）askretrycancel()

askretrycancel()函数在对话框中显示“重试”和“取消”按钮，其返回值分别为 True

或 False，如图 4.18 所示。

例如：

```
a = tkinter.messagebox.askretrycancel('提示', '要执行此操作吗')
print(a)
```

当单击对话框的“重试”按钮时，程序结果为 True；当单击对话框的“取消”按钮时，程序结果为 False。

（4）askyesnocancel()

askyesnocancel()函数在对话框中显示“是”“否”“取消”三个按钮，其返回值分别为 yes、no 或 None，如图 4.19 所示。



图 4.18　消息框中显示“重试”和“取消”按钮



图 4.19　消息框中显示“是”“否”“取消”三个按钮

例如：

```
a = tkinter.messagebox.askretrycancel('提示', '要执行此操作吗')
print(a)
```

当单击对话框的“是”按钮时，程序结果为 True；当单击对话框的“否”按钮时，程序结果为 False；当单击对话框的“取消”按钮时，程序结果为 None。

【例 4-12】 有返回值的消息对话框示例。

```
import tkinter
import tkinter.messagebox
def but_okcancel():
    a = tkinter.messagebox.askokcancel('提示', '要执行此操作吗')
    print(a)

def but_askquestion():
    a = tkinter.messagebox.askquestion('提示', '要执行此操作吗')
    print(a)

def but_trycancel():
    a = tkinter.messagebox.askretrycancel('提示', '要执行此操作吗')
    print(a)

def but_yesnocancel():
   a = tkinter.messagebox.askyesnocancel('提示', '要执行此操作吗')
   print(a)
```

```
root=tkinter.Tk()
root.title('消息对话框')            # 标题
root.geometry('400×400')          # 窗体大小
root.resizable(False, False)      # 固定窗体
tkinter.Button(root, text='确定/取消对话框',\
                     command=but_okcancel).pack()
tkinter.Button(root, text='是/否对话框',\
                     command=but_askquestion).pack()
tkinter.Button(root, text='重试/取消对话框',\
                     command=but_trycancel).pack()
tkinter.Button(root, text='是/否/取消对话框',\
                     command=but_yesnocancel).pack()
root.mainloop()
```

3. 文件对话框 filedialog

（1）导入文件对话框模块语句

```
import tkinter.filedialog
```

（2）获取文件对话框返回值

文件对话框的返回值为文件路径和文件名。

【例 4-13】 文件对话框 filedialog 应用示例。

程序代码如下：

```
import tkinter.filedialog

a = tkinter.filedialog.askopenfilename()
print(a)
```

程序运行结果如图 4.20 所示。

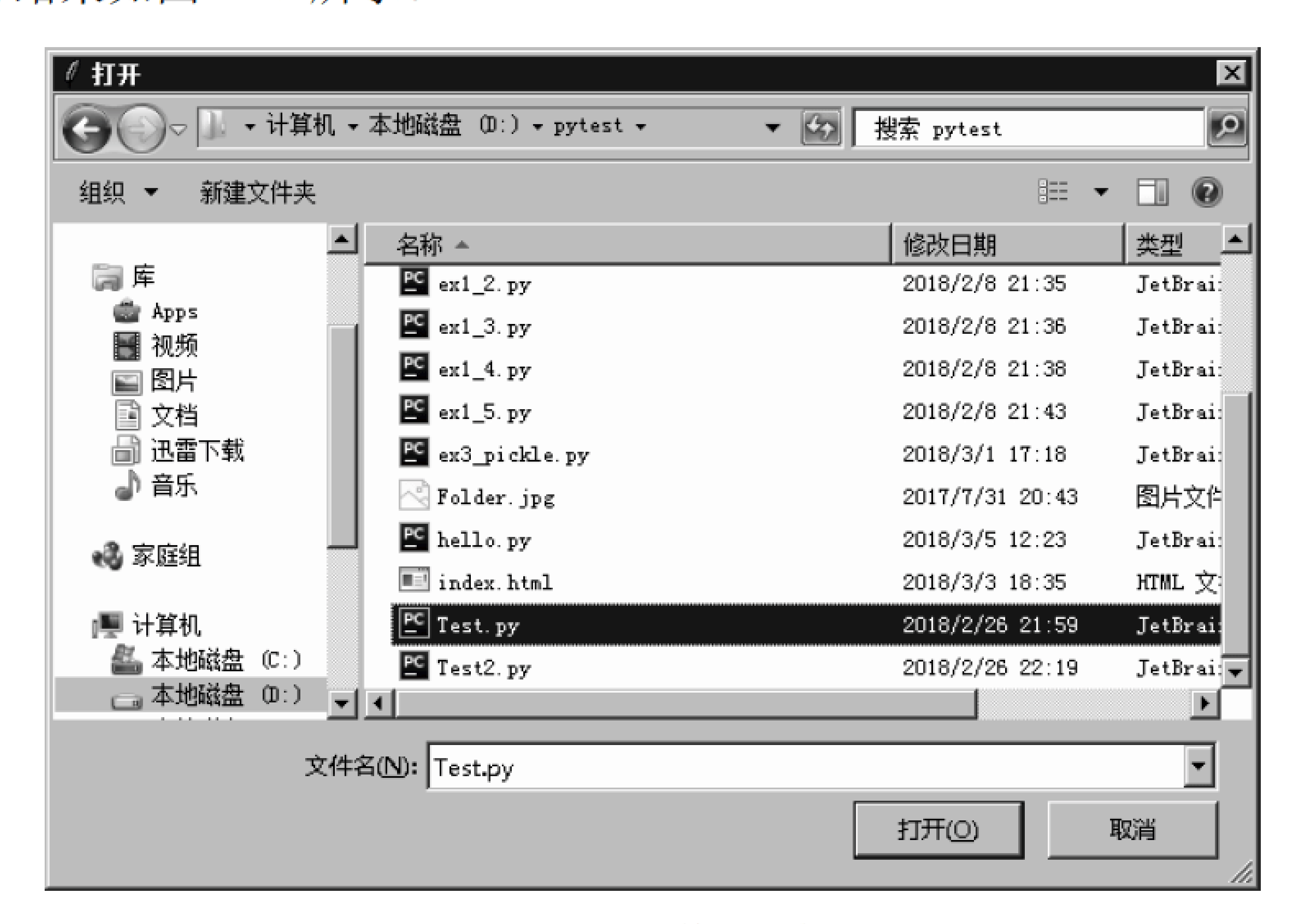


图 4.20 文件对话框

4. 颜色选择对话框 colorchooser

colorchooser.askcolor()提供一个用户选择颜色的界面。其返回值是一个二元组，第 1 个元素是选择的 RGB 颜色值；第 2 个元素是对应的十六进制颜色值。

【例 4-14】 颜色选择对话框示例。

程序代码如下：

```
import tkinter.colorchooser
from tkinter import *

a = colorchooser.askcolor()
print(a)
```

程序运行结果，如图 4.21 所示。选择颜色后，单击“确定”按钮，结果为：

```
((128, 255, 255), '#80ffff')
```

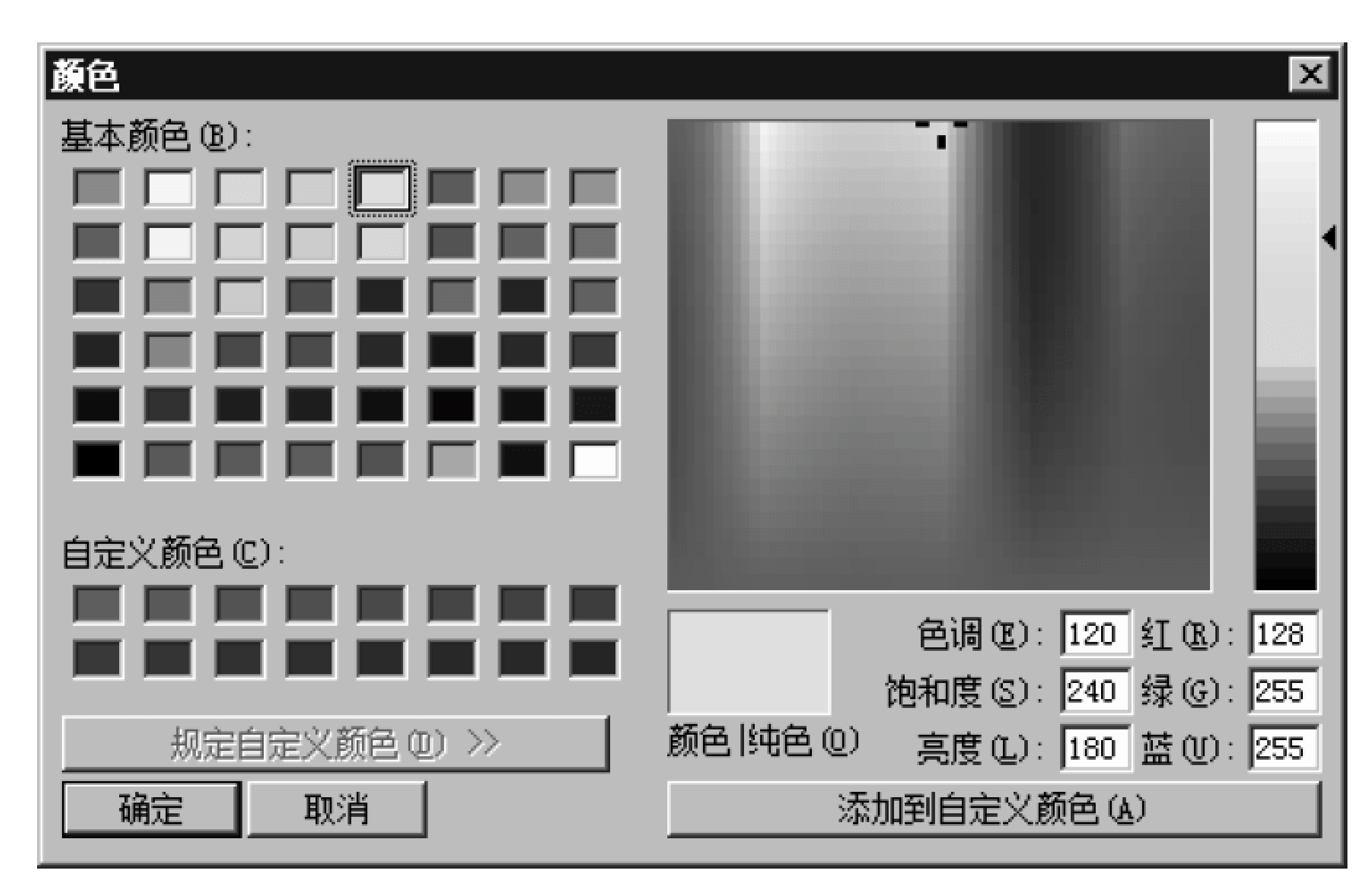


图 4.21 颜色选择对话框


视频录像

4.7 鼠标键盘事件

4.7.1 鼠标事件

在 Python 中，tkinter 模块的事件 event 都用字符串描述，格式为：

```
组件对象.bind(event, handler)
```

其中，event 为事件；handler 为处理事件的函数。

鼠标按钮的点击事件的一般格式为：

```
<ButtonPress-n>
```

其中，n 为鼠标按钮，n 为 1 代表左键；2 代表中键；3 代表右键。

例如，<ButtonPress-1>，表示按下鼠标的左键。

Python 中，定义的鼠标事件如表 4.6 所示。

表 4.6　鼠标事件

事件	说明
<ButtonPress-n>	鼠标按钮 n 被按下，n 为 1 代表左键，2 代表中键，3 代表右键
<ButtonRelease-n>	鼠标按钮 n 被松开
<Bn-Motion>	在按住鼠标按钮 n 的同时，移动鼠标
<Enter>	鼠标进入组件
<Leave>	鼠标离开组件

可以通过鼠标事件 event 获得鼠标位置。坐标点（event.x，event.y）为发生事件时，鼠标所在的位置。

【例 4-15】 编写捕获鼠标点击事件的程序。当鼠标在窗体容器中点击时，记录下其坐标位置。

程序代码如下：

```
from tkinter import *

def callback(event):
    print("clicked at:", event.x, event.y)
    s = (event.x, event.y)
    txt.set(s)

win = Tk()
win.geometry('200×120')
win.title('鼠标事件')

frame = Frame(win, width=200, height=100, bg = 'cyan')
frame.bind("<Button-1>", callback)
frame.pack()

txt =  StringVar()
L = Label(win, width=20, textvariable = txt)
L.pack()
win.mainloop()
```

程序运行结果如图 4.22 所示。



图 4.22　记录单击鼠标的坐标位置

4.7.2 键盘事件

在 Python 中，定义的键盘事件如表 4.7 所示。

表 4.7 键盘事件

事件	说明
<KeyPress>	按下任意的键
<KeyRelease>	松开任意键
<KeyPress-key>	按下指定的 key 键
<KeyRelease-key>	松开指定的 key 键
<Prefix-key>	在按住 prefix 的同时，按下指定的 key 键。其中 prefix 项是 Alt、Shift、Control 中的一项，也可以是它们的组合，如<Control-Alt-key>

在捕获键盘事件时，先要用 focus_set()方法把键盘的焦点设置到一个组件上，这样才能捕获到键盘事件。

几个方向键的键值如表 4.8 所示。

表 4.8 方向键的键值表

方向键	键值描述		方向键	键值描述	
Up（向上）	keysym=Up	keycode=38	Left（向左）	keysym=Left	keycode=37
Down（向下）	keysym=Down	keycode=40	Right（向右）	keysym=Right	keycode=39

【例 4-16】 通过捕获键盘事件，在窗体中显示按下的键。

程序代码如下：

```
from tkinter import *

win = Tk()
win.title('键盘事件')

def key_action(event):
    print("pressed", repr(event.char))
    s = event.char
    txt.set(s)

def callback(event):
    L.focus_set()  ← 把键盘焦点设置到文本标签上

txt =  StringVar()
L = Label(win, width=20, textvariable = txt, font = 'song-36 bold',bg =
'cyan')
L.bind("<KeyPress>", key_action)
L.bind("<Button-1>", callback)
L.pack()
```

```
win.mainloop()
```

程序运行结果如图 4.23 所示。



图 4.23 捕获键盘事件

4.8 案例精选


视频录像

【例 4-17】 设计一个具有加减乘除功能的简单计算器。

程序代码如下：

```
import tkinter
from tkinter import *

# 创建横条型框架
def frame(root, side):
  f = Frame(root)
  f.pack(side = side, expand = YES, fill = BOTH)
  return f

# 统一定义按钮样式和风格
def button(root, side, text, command = None):
  btn = Button(root, text = text, font = ('宋体','12'), command = command)
  btn.pack(side = side, expand = YES, fill = BOTH)
  return btn

# 继承了Frame类，初始化程序界面的布局
class Calculator(Frame):
  def __init__(self):
    Frame.__init__(self)
    self.pack(expand = YES, fill = BOTH)
    self.master.title('简易计算器')
    display = StringVar()

    # 添加显示数字结果的文本框
    Entry(self, relief = SUNKEN, font = ('宋体','20','bold'),\
      textvariable = display).pack(side = TOP, expand = YES,\
        fill = BOTH)

   # 添加清除按钮
```

```
    clearF = frame(self, TOP)
    button(clearF,LEFT,'清除', lambda w = display : w.set(''))

    # 添加横条型框架以及里面的按钮
    for key in('123+', '456-', '789*', '.0=/'):
       keyF = frame(self, TOP)
       for char in key:
          if char == '=':
             btn = button(keyF, LEFT, char)
             btn.bind('<ButtonRelease - 1>',\
                      lambda e, s = self, w = display:s.calc(w), '+')
         else:
             btn = button(keyF, LEFT, char,\
                      lambda w = display, c = char:w.set(w.get()+c))

  # 调用eval函数计算表达式的值
  def calc(self, display):
     try:
        display.set(eval(display.get()))
    except:
        display.set("ERROR")

# 程序的入口
if __name__ == '__main__':
 print('ok')
 Calculator().mainloop()
```

lambda 为匿名函数

运行程序结果如图 4.24 所示。

简易计算器			
12*5			
清除			
1	2	3	+
4	5	6	-
7	8	9	*
.	0	=	/

图 4.24　简易计算器

【例 4-18】 编写程序，测试键盘的按键。

编写程序如下：

```
from tkinter import *

win = Tk()
```

```
win.title('键盘事件')

def key_action(event):
    if(event.keysym=='Up'):          # keysym = Up     keycode=38
        print( "pressed: Up")
    if(event.keycode==40):           # keysym = Down    keycode=40
        print( "pressed: Down")
    if(event.keycode==37):           # keysym = Left    keycode=37
        print( "pressed: Left")
    if(event.keysym=='Right'):       # keysym = Right   keycode=39
        print( "pressed: Right")
    s = event
    txt.set(s)

def callback(event):
    L.focus_set()

txt = StringVar()   #courier
L = Label(win, width=70, textvariable = txt, font = 'song -16',bg = 'cyan')
L.bind("<KeyPress>", key_action)
L.bind("<Button-1>", callback)
L.pack()

win.mainloop()
```

运行程序，当按键盘上的键时，在窗体中显示其相应的键值信息，如图 4.25 所示。



图 4.25　在窗体上显示相应的键值

习　题　4

1．创建一个窗体，窗体中有一个按钮，当单击该按钮后，就会弹出一个新窗体。

2．设计一个加法计算器，如图 4.26 所示。在文本框中输入两个整数，单击“＝”按钮时，在第三个文本框中显示这两个数的和。

3．编写程序包含一个标签、一个文本框和一个按钮，当用户单击按钮时，程序把文本框中的内容复制到标签中。

4．设计一个类似 Windows 系统的计算器，要求使用按钮、文本框、布局管理、标签等构件，实现多位数的加、减、乘、除运算功能。

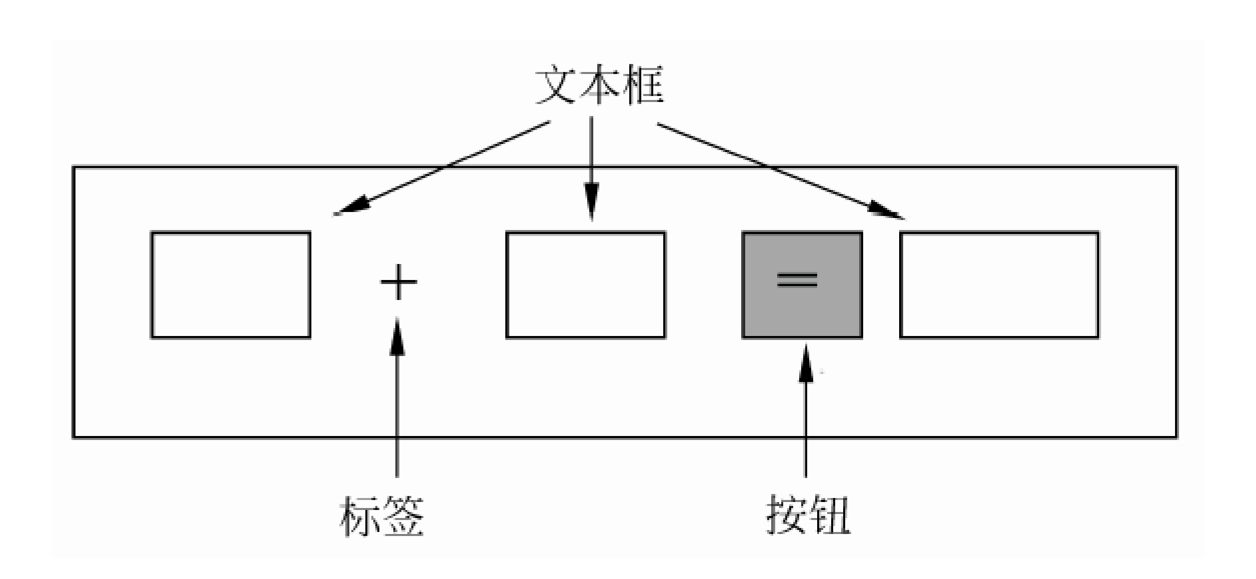


图 4.26　加法计算器

5．编写图形界面的应用程序，该程序包含一个菜单，选择这个菜单的“退出”选项可以关闭窗口并结束程序。

6．设计一个模拟的文字编辑器，并用菜单实现退出的功能。

第 5 章　绘图及图像处理

5.1　绘 制 图 形



视频录像

5.1.1　用画布组件绘图

画布是图形用户界面 tkinter 的组件，是一个矩形区域，用于绘制图形或作为容器放置其他组件。

1. 创建画布对象

创建画布对象的基本语法形式如下：

```
w = Canvas(master, option=value, … )
```

其中：

- master：代表父窗口。
- options：为属性参数，其意义如表 5.1 所示。

表 5.1　画布的常用参数

Option 参数	说明
bg	背景颜色
height	画布的高
width	画布的宽

2. 图形的绘制方法

Canvas 对象包含了大量的绘图方法，表 5.2 列出了常用的绘图方法。

表 5.2　Canvas 对象常用的绘图方法

方法	说明
create_line(x1, y1, x2, y2)	绘制一条从(x1,y1)到(x2,y2)的直线
create_rectangle(x1, y1, x2, y2)	绘制一个左上角为(x1,y1)，右下角为(x2,y2)的矩形
create_polygon(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6)	绘制一个顶点为(x1,y1)，(x2,y2),…,的多边形
create_oval(x1, y1, x2, y2, fill='color')	绘制一个左上角为(x1,y1)，右下角为(x2,y2)的外接矩形包围的圆，fill 为填充颜色
create_arc(x1, y1, x2, y2, start=s0, extent=s)	绘制在左上角为(x1,y1)，右下角为(x2,y2)的外接矩形所包围的一段圆弧，圆弧角度为 s，从 s0 开始
create_image(w, h, anchor=NE, image=filename)	在 w 宽 h 高的矩形区域内，显示文件名为 filename 的图像
move(obj, x, y)	移动组件 obj。x 为水平方向变化量，y 为垂直方向变化量

【例 5-1】 绘制几何图形示例。

程序代码如下：

```
'''
    窗体中的画布示例：
     绘制小球和扇形
'''
import tkinter
import tkinter.messagebox
win = tkinter.Tk()
win.title('画布示例')          # 定义窗体标题
win.geometry('400×200')       # 定义窗体的大小400×200像素
can = tkinter.Canvas(win, height=200, width=400)         # 定义画布
id = can.create_line(15,15,190,15)                       # 画一条直线
io1 = can.create_oval(50, 50, 100, 100, fill='blue')     # 画一蓝色圆
io2 = can.create_oval(59, 59, 68, 68, fill='white')      # 画一白色小圆
coord = 15, 120, 210, 220
arc = can.create_arc(coord, extent=150, fill="green")    # 画一个扇形
can.pack()
win.mainloop()
```

程序运行结果如图 5.1 所示。

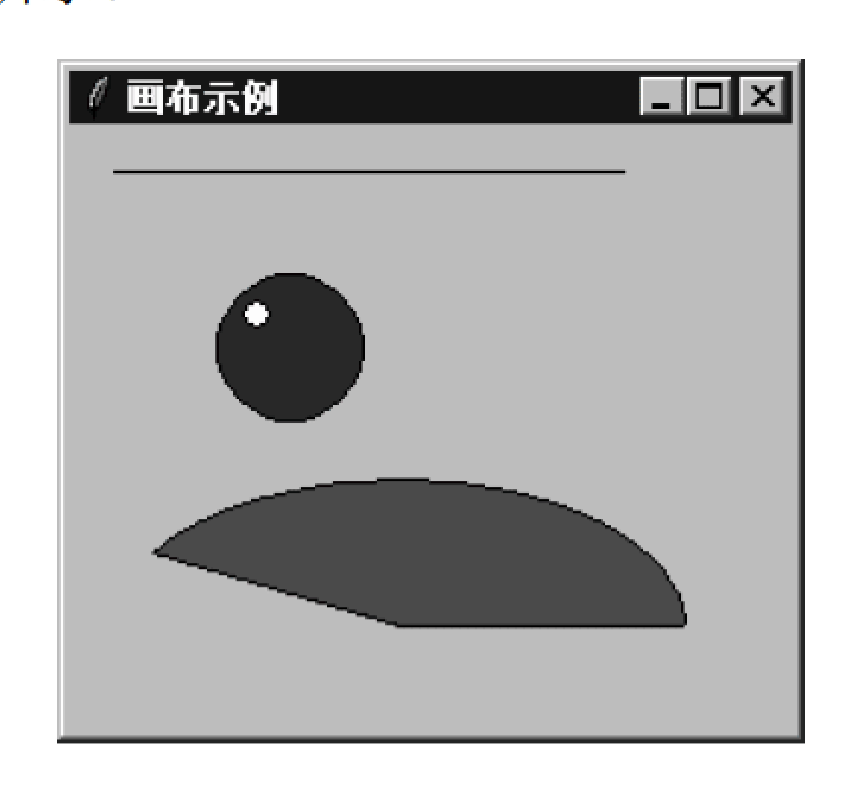


图 5.1　绘制几何图形

【例 5-2】 绘制笑脸。

程序代码如下：

```
'''
    窗体中的画布示例：
     绘制笑脸
'''
import tkinter
import tkinter.messagebox
win = tkinter.Tk()
win.title('画布示例')
```

```
win.geometry('250×250')

can = tkinter.Canvas(win, height=250, width=250)        # 定义画布
io1 = can.create_oval(35,30,210,210, fill='yellow')     # 画一黄色圆
io2 = can.create_oval(70,70,180,180, fill='black')
io3 = can.create_oval(65,70,185,170, outline='yellow', fill='yellow')
io4 = can.create_oval(80,100,110,130, fill='black')
io5 = can.create_oval(150,100,180,130, fill='black')

can.pack()
win.mainloop()
```

程序运行结果如图 5.2 所示。

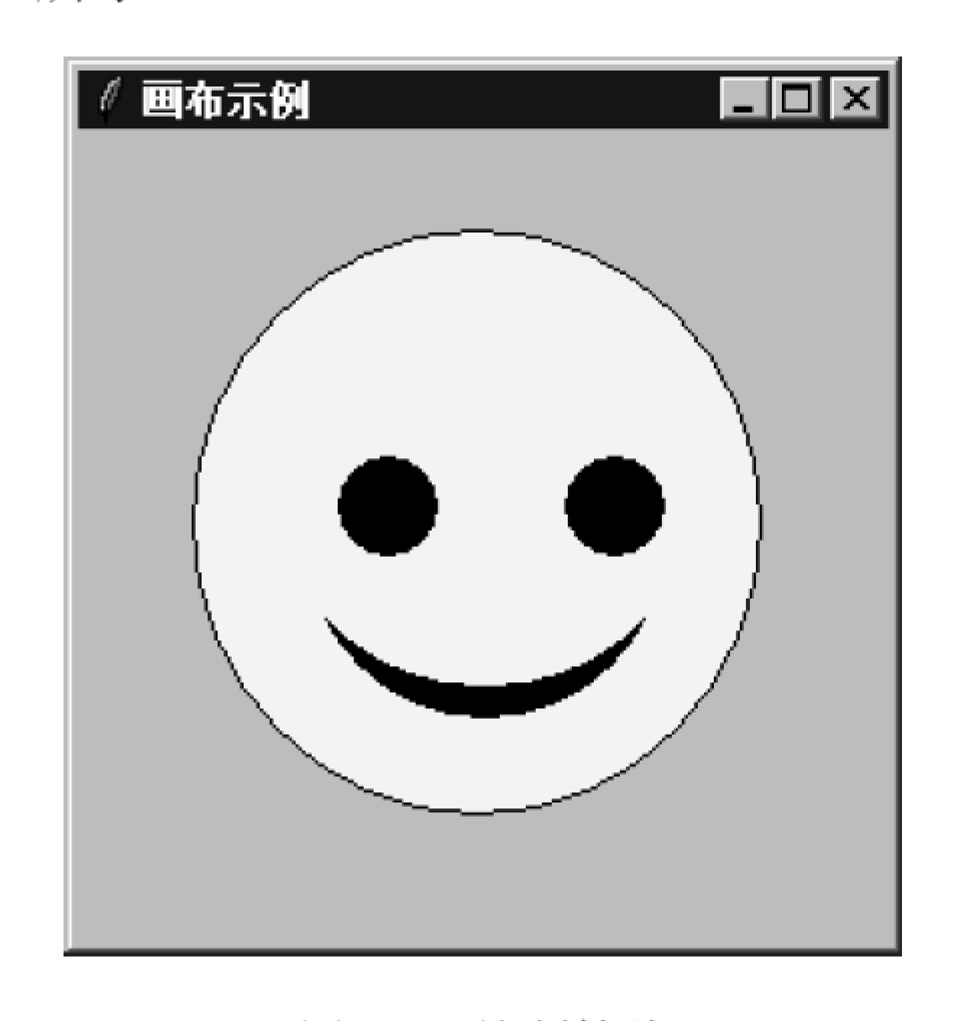


图 5.2　绘制笑脸

【例 5-3】 显示图像示例。

程序代码如下：

```
import tkinter.messagebox
from tkinter import *

win = tkinter.Tk()
win.title('绘图示例')          # 定义窗体标题
win.geometry('200×200')       # 定义窗体的大小200×200像素
can = tkinter.Canvas(win, height=200, width=200)    # 定义画布
filename = PhotoImage(file = "test.gif")
image = can.create_image(150, 10, anchor=NE, image=filename)
can.pack()
win.mainloop()
```

程序运行结果如图 5.3 所示。



图 5.3　显示图像

5.1.2　用 turtle 模块绘图

turtle 模块是 Python 中的一个简单绘图工具，用它绘图非常方便。使用 turtle 绘制图形时，它会显示出一个箭头（又称为“海龟”），该箭头在一个横轴为 x、纵轴为 y 的坐标系中，从原点(0, 0)位置开始，按照所绘图形的轨迹绘制图形。

下面介绍 turtle 模块的一些基础知识。

1. turtle 模块的画布 Canvas

画布 Canvas 是 turtle 用于绘图区域，可以设置它的大小和初始位置。

（1）设置画布大小

```
turtle.screensize(canvwidth=None,canvheight=None,bg=None)
```

其中，参数 canvwidth 为画布的宽(单位像素)；canvheight 为高；bg 为背景颜色。

例如：

```
turtle.screensize(800, 600, "green")
```

当 screensize()函数无参数时，则返回一个默认为宽 400，高 300 像素的画布

即

```
turtle.screensize() # 返回默认大小(400,300)
```

（2）设置画布初始位置

```
turtle.setup(width=0.5,height=0.75,startx=None,starty=None)
```

其中参数：

width,height：当宽和高为整数时，表示像素；为小数时，表示占据屏幕的比例。

(startx,starty)：表示矩形窗口左上角顶点的坐标位置，如果为空，则位于屏幕中心。

例如：

```
turtle.setup(width=800,height=800,startx=100,starty=100)
turtle.setup(width=0.6,height=0.6)   # 画布位于屏幕中心
```

2. turtle 模块的基本指令

操纵 turtle 模块的“海龟”绘图有许多命令，这些命令分为两种：一种为画笔控制命

令；另一种为运动命令。

（1）画笔控制命令

turtle 模块的画笔控制命令如表 5.3 所示。

表 5.3　画笔控制命令

画笔控制命令	说明
turtle.down()	画笔落下，移动时绘制图形
turtle.up()	画笔抬起，移动时不绘制图形
turtle.pensize(width)	设置画笔的宽度，即绘制图形线条的宽度
turtle.color(colorstring)	设置画笔的颜色，即绘制图形的颜色
turtle.fillcolor(colorstring)	设置绘制图形的填充颜色
turtle.fill(true)	绘制填充图形
turtle.fill(false)	绘制线条图形
turtle. circle(radius, extent)	绘制一个圆形，其中 radius 为半径；extent 为角度。例如，若 extent 为 180，则画一个半圆；如画一个圆形，则不必写第二个参数

（2）运行命令

turtle 模块的运行命令如表 5.4 所示。

表 5.4　运行命令

运动命令	说明
turtle.forward(d)	向前移动距离，d 代表距离
turtle.backward(d)	向后移动距离，d 代表距离
turtle.right(degree)	向右转动多少角度
turtle.left(degree)	向左转动多少角度
turtle.goto(x,y)	将画笔移动到坐标为(x,y)的位置
turtle.stamp()	绘制当前图形
turtle.speed(speed)	画笔绘制的速度，取值范围为[0,10]的整数，值越大速度越快
turtle.clear()	清空 turtle 画的笔迹
turtle.reset()	清空窗口，重置 turtle 的状态为起始状态
turtle.undo()	撤销上一个 turtle 动作
turtle.isvisible()	设置当前 turtle 是否可见
turtle.stamp()	复制当前图形
turtle.write('str')	写字符串'str'
turtle.write(str[, font=("font-name", font_size,"font_type")])	写文本，str 为文本内容，font 是字体的参数，里面分别为字体名称、大小和类型；font 为可选项，font 的参数也是可选项

【例 5-4】 绘制一个边长为 60 的三角形图形。

程序代码如下：

```
import turtle
import time
a=60

for n in range(1, 4):
```

```
        turtle.forward(a)
        turtle.left(120)
        turtle.speed(1)
    time.sleep(5)
```

程序运行结果如图 5.4 所示。

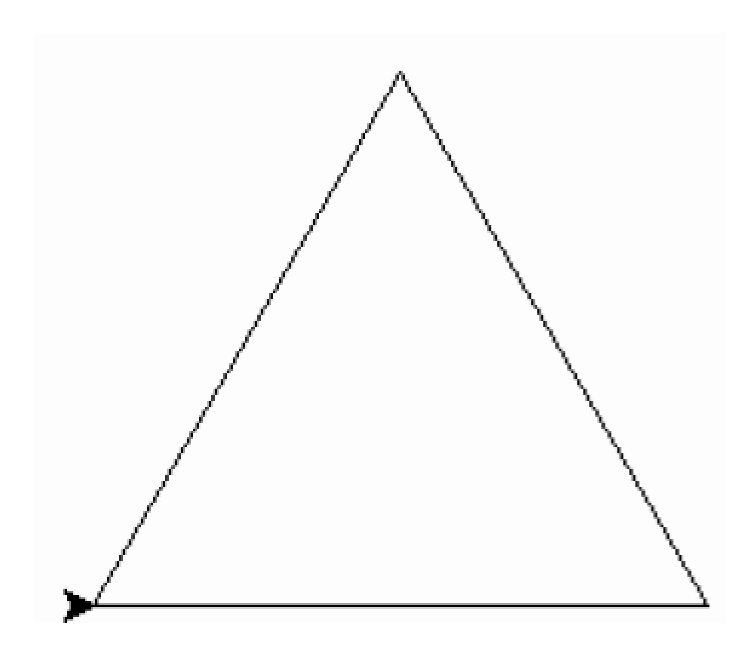


图 5.4　绘制三角形图形

5.2　数字图像处理基础



视频录像

5.2.1　Python 图像处理类库 PIL

图像处理类库（Python Imaging Library，PIL）提供了通用的图像处理功能，以及大量实用的基本图像操作，如图像缩放、裁剪、旋转、颜色转换等。由于 PIL 仅支持 Python 2.7 以前版本，Python 3.x 的兼容版本称为 Pillow。

1. 安装 Pillow 模块

在命令行窗口中使用 pip 安装 Pillow 模块，其命令为：

```
pip install pillow
```

安装过程如图 5.5 所示。

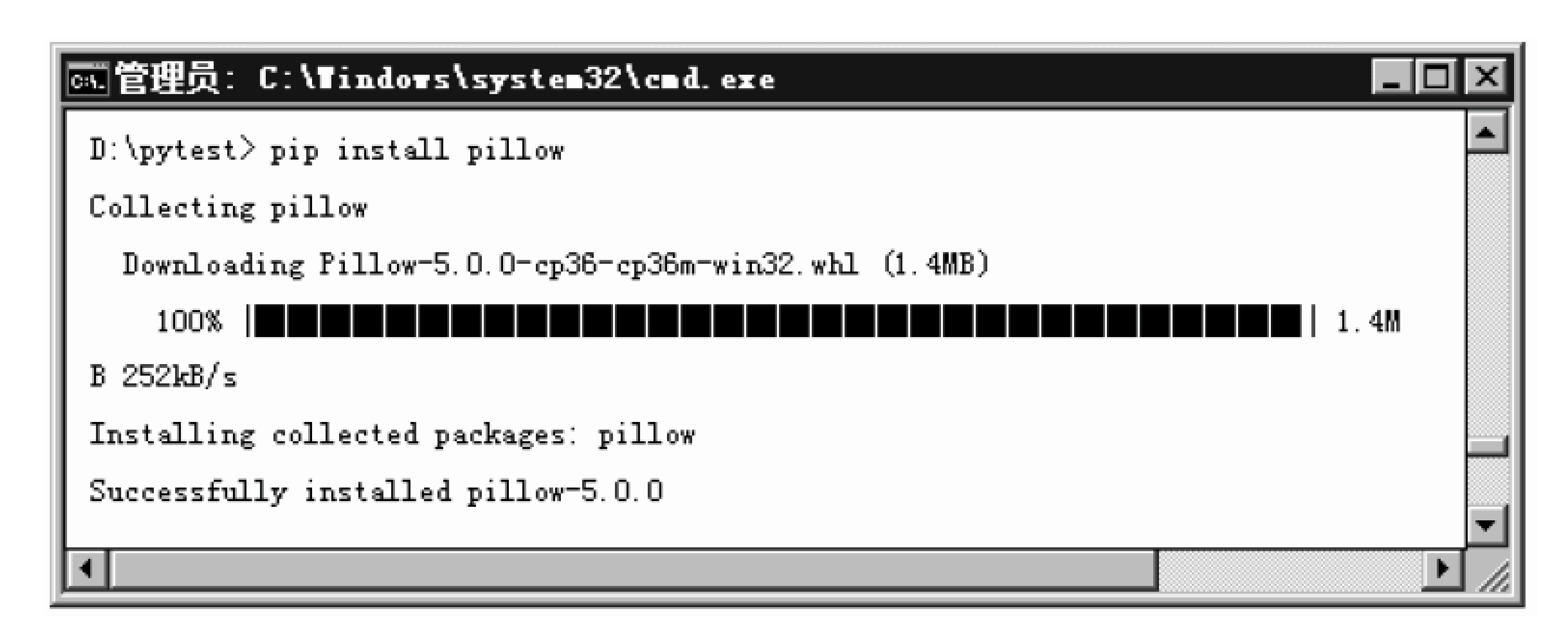


图 5.5　安装 Pillow 模块

2. Pillow 模块的方法

Pillow 模块提供了大量用于图像处理的方法，通过创建的图像对象可以调用这些图像

处理方法。Pillow 模块图像处理的常用方法如表 5.5 所示。

表 5.5 Pillow 模块图像处理的常用方法

方法	说明
Image.open("图像文件名")	打开图像文件，返回图像对象
show()	显示图像
save("文件名")	保存图像文件
resize(宽高元组)	图像缩放
thumbnail()	创建图像的缩略图
rotate()	旋转图像
transpose(Image.FLIP_LEFT_RIGHT)	图像水平翻转
transpose(Image.FLIP_TOP_BOTTOM)	图像垂直翻转
crop(矩形区域元组)	裁剪图像
paste(裁剪图像对象，矩形区域)	粘贴图像
ImageGrab.grab(矩形区域元组)	屏幕截图，若区域为空，则表示全屏幕截图
filter(ImageFilter.EDGE_ENHANCE)	图像增强
filter(ImageFilter.BLUR)	图像模糊
filter(ImageFilter.FIND_EDGES)	图像边缘提取
point(lambda i:i*r)	图像点运算。r>1，图像变亮；r<1,图像变暗
format	查看图像格式的属性值
size	查看图像大小的属性值，格式为（宽度，高度）
getpixel(坐标元组)	读取像素的属性值，参数为(x,y)坐标元组，返回值为红、绿、蓝三色分量的值
putpixel((元组 1), (元组 2))	元组 2 的值改变目标像素元组 1 的颜色值
split()	将彩色图像分离为红、绿、蓝三个分量通道。 例如：r, g, b = im.split()
Image.merge(im.mode, (r,g,b))	将红、绿、蓝三个分量通道合并成一个彩色图像
enhance(n)	对比度增强为原来的 n 倍(n 为实数)。例如： img = ImageEnhance.Contrast(img) img = im.enhance(1.5) #对比度增强为原图的 1.5 倍

5.2.2 图像处理技术

利用 PIL 中的函数，可以从大多数图像格式的文件中读取数据，然后写入最常见的图像格式文件中。PIL 中最重要的模块为 Image。要读取一幅图像，可以使用下列语句：

```
from PIL import Image
img = Image.open("img1.gif")
```

上述代码的返回值 pil_im 是一个 PIL 图像对象。可以对这个 PIL 图像对象进行各种处理。下面介绍几个典型的图像处理的应用示例。

1. 图像的打开、旋转和显示

【例 5-5】 打开和显示图像示例。

程序代码如下：

```
import tkinter
```

```
from PIL import Image, ImageTk

win = tkinter.Tk()
win.title('图像显示')
win.geometry('300×300')                # 定义窗体的大小300×300像素

can = tkinter.Canvas(win,              # 创建画布组件
    bg='white',                        # 指定画布组件的背景色
    width=300,                         # 指定画布组件的宽度
    height=300)                        # 指定画布组件的高度

image = Image.open("dukou.jpg")        # 打开图像文件
img = ImageTk.PhotoImage(image)        # 获取图像像素
can.create_image(160,120,image=img)    # 将图像添加到画布组件中
can.pack()                             # 将画布组件添加到主窗口

win.mainloop()
```

程序运行结果如图 5.6 所示。



图 5.6　打开和显示图像

2. 建立图像的缩略图

使用 PIL 可以方便地创建图像的缩略图。PIL 图像对象的 thumbnail(size)方法将图像转换成由元组参数设定大小的缩略图。

【例 5-6】 建立图像缩略图示例。

程序代码如下：

```
import tkinter
from tkinter import Label
from PIL import Image, ImageTk
import glob, os
```

```
win = tkinter.Tk()
win.title('建立图像缩略图')
win.geometry('200×200')                          # 定义窗体的大小400×200像素

def imgshow():
    size = 64, 64                                # 设置缩略图尺寸的元组参数
    for infile in glob.glob("dukou.jpg"):
        file, ext = os.path.splitext(infile)
        im = Image.open(infile)
        im.thumbnail(size)
        im.save(file + "(1).jpg", "JPEG")    # 保存缩略图为dukou(1).jpg
    photo = ImageTk.PhotoImage(file='dukou(1).jpg')
    label = Label(win, image=photo).pack()
    label.image = photo

tkinter.Button(win, text='建立图像缩略图 ',command=imgshow).pack()
win.mainloop()
```

运行程序，单击按钮后，将当前文件夹中名为 dukou.jpg 的图像文件生成 64×64 像素的缩略图，如图 5.7 所示。



图 5.7　生成图像缩略图

3. 增强图像处理

使用 PIL 模块可以方便地对图像进行各种处理。例如，应用 filter()方法的 ImageFilter.EDGE_ENHANCE 属性可以将图像的对比度增强。

【例 5-7】 增加图像的对比度示例。

程序代码如下：

```
import tkinter
from tkinter import Label
from PIL import Image, ImageTk, ImageEnhance, ImageFilter

win = tkinter.Tk()
win.title('增强图像')
win.geometry('400×200')                 # 定义窗体的大小400×200像素
```

```
photo = Image.open('dukou.jpg')
img1 = ImageTk.PhotoImage(photo)            # 获取图片像素
label_1 = Label(win, image=img1)            # 显示原图

def imgshow():
    img = photo.filter(ImageFilter.EDGE_ENHANCE)
    img2 = ImageTk.PhotoImage(img)          # 获取图片像素
    label_2 = Label(win, image=img2).grid(row=1, column=1)  # 显示增强后的图
    label_2.image = img2

button = tkinter.Button(win, text='增强图像处理 ',command=imgshow)

button.grid(row=0, column=0, columnspan=2)
label_1.grid(row=1, column=0)

win.mainloop()
```

程序运行结果如图 5.8 所示。



图 5.8　图像增强

5.3　案 例 精 选



视频录像

【例 5-8】 动画效果的签名。

程序代码如下：

```
import turtle

turtle.color('red','green')
turtle.pensize(5)
turtle.goto(0,0)
turtle.speed(10)
for i in range(15):
    turtle.forward(100)
```

```
    turtle.right(150)
    turtle.up()

turtle.goto(100,-120)
turtle.color('black')
turtle.write("Python爱好者",font="隶书 -36 bold")
turtle.up()

turtle.goto(135,-140)
turtle.color('black')
turtle.write("2018 年 1 月 1 日",font="隶书 -18" )
turtle.up()
turtle.goto(240,-160)
turtle.color('black')
turtle.write("." )
turtle.done()
```

程序运行结果如图 5.9 所示。



图 5.9　绘制有动画效果的签名

【例 5-9】 绘制一个指针式时钟。

程序代码如下：

```
import turtle
from datetime import *
# 抬起画笔，向前运动一段距离放下
def Skip(step):
  turtle.penup()
  turtle.forward(step)
  turtle.pendown()
def mkHand(name, length):
  # 注册Turtle形状，建立表针Turtle
  turtle.reset()
  Skip(-length * 0.1)
  # 开始记录多边形的顶点。当前的乌龟位置是多边形的第一个顶点
  turtle.begin_poly()
  turtle.forward(length * 1.1)
```

```
    # 停止记录多边形的顶点。当前的乌龟位置是多边形的最后一个顶点。将最后一个顶点与第一个
    # 顶点相连
    turtle.end_poly()
    # 返回最后记录的多边形
    handForm = turtle.get_poly()
    turtle.register_shape(name, handForm)
def Init():
    global secHand, minHand, hurHand, printer
    # 重置Turtle指向北
    turtle.mode("logo")
    # 建立三个表针Turtle并初始化
    mkHand("secHand", 135)
    mkHand("minHand", 125)
    mkHand("hurHand", 90)
    secHand = turtle.Turtle()
    secHand.shape("secHand")
    minHand = turtle.Turtle()
    minHand.shape("minHand")
    hurHand = turtle.Turtle()
    hurHand.shape("hurHand")
    for hand in secHand, minHand, hurHand:
        hand.shapesize(1, 1, 3)
        hand.speed(0)
    # 建立输出文字Turtle
    printer = turtle.Turtle()
    # 隐藏画笔的turtle形状
    printer.hideturtle()
    printer.penup()
def SetupClock(radius):
    # 建立表的外框
    turtle.reset()
    turtle.pensize(7)
    for i in range(60):
        Skip(radius)
        if i % 5 == 0:
            turtle.forward(20)
            Skip(-radius - 20)

            Skip(radius + 20)
            if i == 0:
                turtle.write(int(12), align="center", font=("Courier", 14, "bold"))
            elif i == 30:
                Skip(25)
                turtle.write(int(i/5), align="center", font=("Courier", 14, "bold"))
```

```
        Skip(-25)
      elif (i == 25 or i == 35):
        Skip(20)
        turtle.write(int(i/5), align="center", font=("Courier", 14, "bold"))
        Skip(-20)
      else:
        turtle.write(int(i/5), align="center", font=("Courier", 14, "bold"))
        Skip(-radius - 20)
    else:
      turtle.dot(5)
      Skip(-radius)
    turtle.right(6)
def Week(t):
  week = ["星期一", "星期二", "星期三", \
      "星期四", "星期五", "星期六", "星期日"]
  return week[t.weekday()]
def Date(t):
  y = t.year
  m = t.month
  d = t.day
  return "%s %d%d" % (y, m, d)
def Tick():
  # 绘制表针的动态显示
  t = datetime.today()
  second = t.second + t.microsecond * 0.000001
  minute = t.minute + second/60.0
  hour = t.hour + minute/60.0
  secHand.setheading(6 * second)
  minHand.setheading(6 * minute)
  hurHand.setheading(30 * hour)
  turtle.tracer(False)
  printer.forward(65)
  printer.write(Week(t), align="center",
         font=("Courier", 14, "bold"))
  printer.back(130)
  printer.write(Date(t), align="center",
         font=("Courier", 14, "bold"))
  printer.home()
  turtle.tracer(True)
  # 100ms后继续调用tick
  turtle.ontimer(Tick, 100)
def main():
  # 打开/关闭龟动画，并为更新图纸设置延迟
```

```
  turtle.tracer(False)
  Init()
  SetupClock(160)
  turtle.tracer(True)
  Tick()
  turtle.mainloop()

if __name__ == "__main__":
  main()
```

程序运行结果如图 5.10 所示。

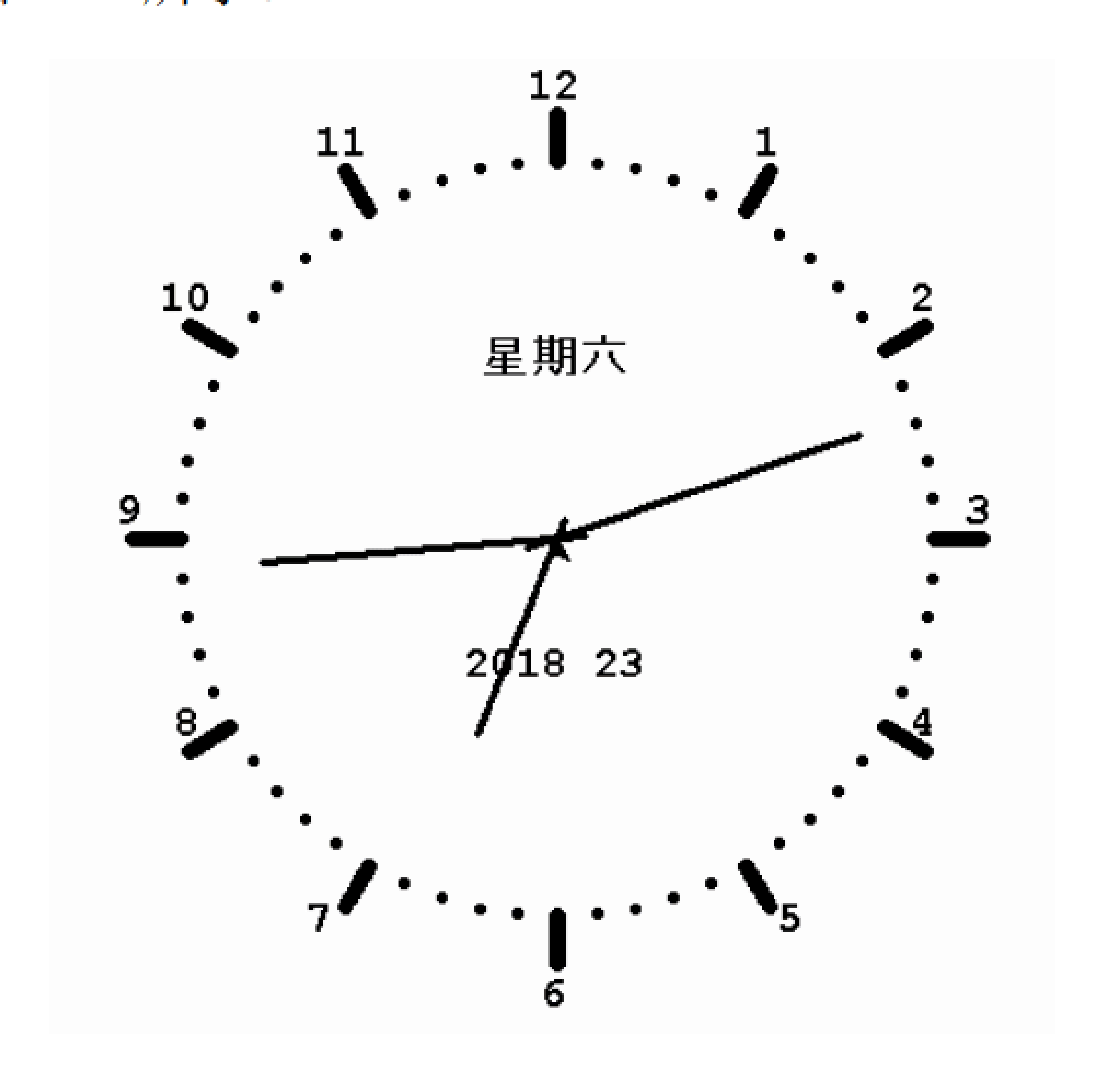


图 5.10　指针式时钟

画布 Canvas 类可以用于设计动画，使用 move(tags，dx，dy)方法实现移动图片或文字等组件。

canvas.update()为刷新界面，重新显示画布。

【例 5-10】 用方向键移动小矩形块。

程序代码如下：

```
import time
from tkinter import *

x = 50
y = 50

# （1）定义窗口
win = Tk()
win.title("移动小矩形块")
# （2）定义画布
```

```
canvas = Canvas(win, width = 400, height = 400)
canvas.pack() # 显示画布

# （3）定义矩形块
rect = canvas.create_rectangle(x, y, x+30, y+30, fill='red')
print(rect)

# （4）定义移动小矩形的函数
def moveRect(event):
    if event.keysym == 'Up':
        canvas.move(rect, 0, -3)
    elif event.keysym == 'Down':
        canvas.move(rect, 0, +3)
    elif event.keysym == 'Left':
        canvas.move(rect, -3, 0)
    elif event.keysym == 'Right':
        canvas.move(rect, 3, 0)
    win.update()          # 界面刷新
    time.sleep(0.05)      # 休眠

# （5）绑定方向键
canvas.bind_all('<KeyPress-Up>', moveRect)
canvas.bind_all('<KeyPress-Down>', moveRect)
canvas.bind_all('<KeyPress-Left>', moveRect)
canvas.bind_all('<KeyPress-Right>', moveRect)

win.mainloop()
```

Keysym == 键值（方向键）
move(组件，x 坐标增量，y 坐标增量)

绑定键盘事件

程序运行结果如图 5.11 所示。

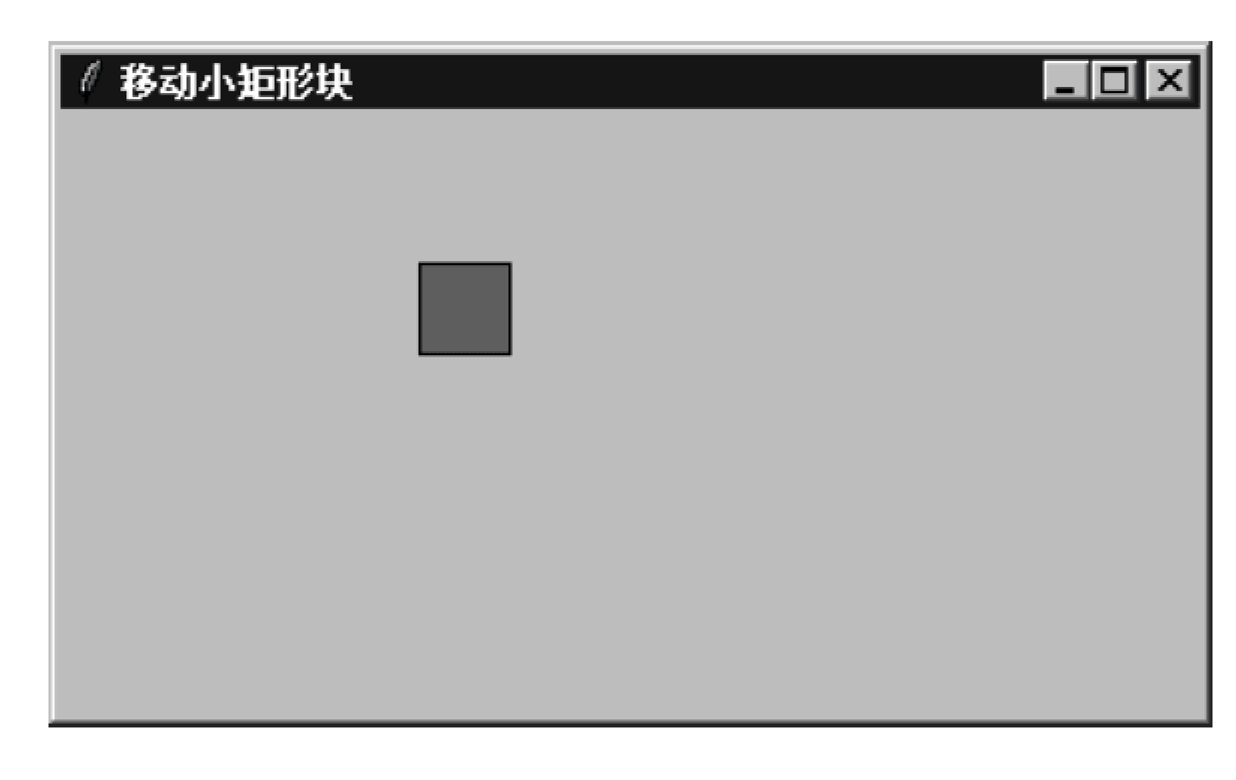


图 5.11　用方向键移动小矩形块

【例 5-11】 设计一个小球遇到窗体边缘或挡板则弹回来的动画程序。

程序代码如下：

```
from tkinter import *
```

```
import random
import time

class Ball:                          # 小球的类
    def __init__(self,canvas,paddle,color):
        self.canvas=canvas           # 传递画布值
        self.paddle=paddle           # 把挡板传递进来
        self.id=canvas.create_oval(10,10,35,35,fill=color)# 画椭圆并保存其ID
        self.canvas.move(self.id,245,100)
        start=[-3,-2,-1,1,2,3]
        random.shuffle(start)        # 随机化列表
        self.x=start[0]
        self.y=-3
        self.canvas_heigh=self.canvas.winfo_height()    # 获取窗口高度并保存
        self.canvas_width=self.canvas.winfo_width()
    def draw(self):
        self.canvas.move(self.id,self.x,self.y)
        # 返回相应ID代表的图形的当前坐标(左上角和右上角坐标)
        pos=self.canvas.coords(self.id)
        # 使得小球不会超出窗口
        pad=self.canvas.coords(self.paddle.id)          # 获取挡板的坐标
        if pos[1]<=0 :
            self.y=3
        if pos[3]>=self.canvas_heigh or(pos[3]>=pad[1] and \
pos[2]>=pad[0] and pos[2]<=pad[2]):
            self.y=-3
        if pos[0]<=0:
            self.x=3
        if pos[2]>=self.canvas_width:
            self.x=-3

class Paddle:  # 挡板的类
    def __init__(self,canvas,color):
        self.canvas=canvas
        self.color=color
        self.id=canvas.create_rectangle(0,0,100,10,fill=color)
        self.canvas.move(self.id,200,300)
        self.canvas_width=self.canvas.winfo_width()
        self.l=0
        self.r=0

    def draw(self):
        pos=self.canvas.coords(self.id)
        if pos[0]<=0:
            self.l=0
```

```
            if pos[2]>=self.canvas_width:
                self.r=0

    def turn_left(self,event):
        self.canvas.move(self.id,self.l,0)
        self.l=-20

    def turn_right(self,event):
        self.canvas.move(self.id,self.r,0)
        self.r=20
tk=Tk()
tk.title('Game')
tk.resizable(0,0)                           # 使得窗口大小不可调整
tk.wm_attributes('-topmost',1)              # 包含画布的窗口放在其他窗口的前面
canvas=Canvas(tk,width=500,height=400,bd=0,highlightthickness=0)
                                            # 后面两个参数去掉边框
canvas.pack()
tk.update()
paddle=Paddle(canvas,'blue')
ball=Ball(canvas,paddle,'red')

canvas.bind_all('<KeyPress-Left>',paddle.turn_left)     # 绑定方向键
canvas.bind_all('<KeyPress-Right>',paddle.turn_right)

while 1:
    ball.draw()
    paddle.draw()
    tk.update_idletasks()                               # 快速重画屏幕
    tk.update()
    time.sleep(0.01)
```

运行程序，红色小球一直处于运行状态，遇到墙壁（窗体边缘）或挡板则按相反路径弹回来，如图 5.12 所示。

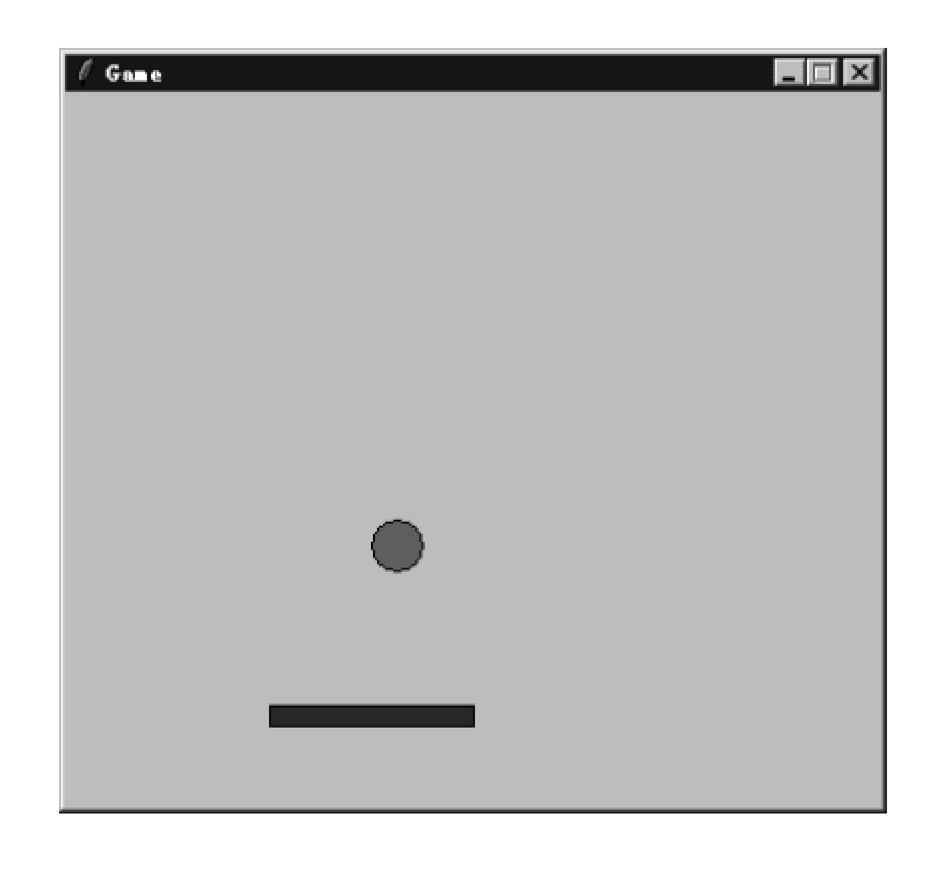


图 5.12　小球碰撞游戏

【例 5-12】 应用图像处理技术，编写一个简易图像处理器。

为了实现更多的图像处理功能，这里安装一个 Python 系统专业绘图库模块 matplotlib。通过 matplotlib，开发者可以仅需要几行代码，便可以方便地进行图像处理，也可以生成绘图、直方图、功率谱、条形图、散点图等。

可以用 pip 安装 matplotlib 模块，其安装命令如下：

```
pip install matplotlib
```

程序代码如下：

```
import tkinter
from tkinter import *
from PIL import Image
import matplotlib.pyplot as plt

# 定义窗体
win = Tk()
win.title("简易图像处理器")

# 定义标题
lab = Label(win, text='简易图像处理器', font=('Times','20','bold'))
lab.grid(row=0, column=0, columnspan=5)

# 定义空白标签
labss = Label(win,text="",width = 50,height = 1)
labss.grid(row=3, column=0, columnspan=5)

# 定义显示图像信息的文本框
s = StringVar()
txt = Entry(win,width=50,font=('宋体','10'),textvariable=s)
txt.grid(row=4, column=0, columnspan=5)

# 定义图像对象
img=Image.open('dukou.gif')
plt.figure("图像处理")

# 显示原像函数
def com_show():
    plt.subplot(2,2,1), plt.title('origin')   # 区域分成1行2列, 第1
    plt.imshow(img)
    plt.axis('off')
    plt.show()

# 查看图像信息函数
def com_info():
```

```
    plt.imshow(img)
    s.set('图片的尺寸:'+str(img.size)+'图片的格式:'+str(img.format))

# 转换灰度函数
def com_gray():
    plt.subplot(2,2,2), plt.title('gray')       # 区域分成1行2列，第2
    gray=img.convert('L')                       # 转换成灰度
    plt.imshow(gray,cmap='gray')
    plt.axis('off')
    plt.show()

# 裁剪图片函数
def com_roi():
    box=(80,100,260,300)
    roi=img.crop(box)
    plt.subplot(2,2,3), plt.title('crop')       # 区域分成1行2列，第3
    plt.imshow(roi),plt.axis('off')
    plt.show()

# 图片左右翻转函数
def com_trans():
    plt.subplot(2,2,4), plt.title('trans')      # 区域分成1行2列，第4
    dst=img.transpose(Image.FLIP_LEFT_RIGHT)    # 左右翻转
    #img.rotate(45)     # 顺时针旋转45°
    plt.imshow(dst)
    plt.axis('off')
    plt.show()

# 定义按钮
btn_show = Button(win, text='显示图像', command=com_show)
btn_show.grid(row=2, column=0)

btn_show = Button(win, text='查看图像信息', command=com_info)
btn_show.grid(row=2, column=1)

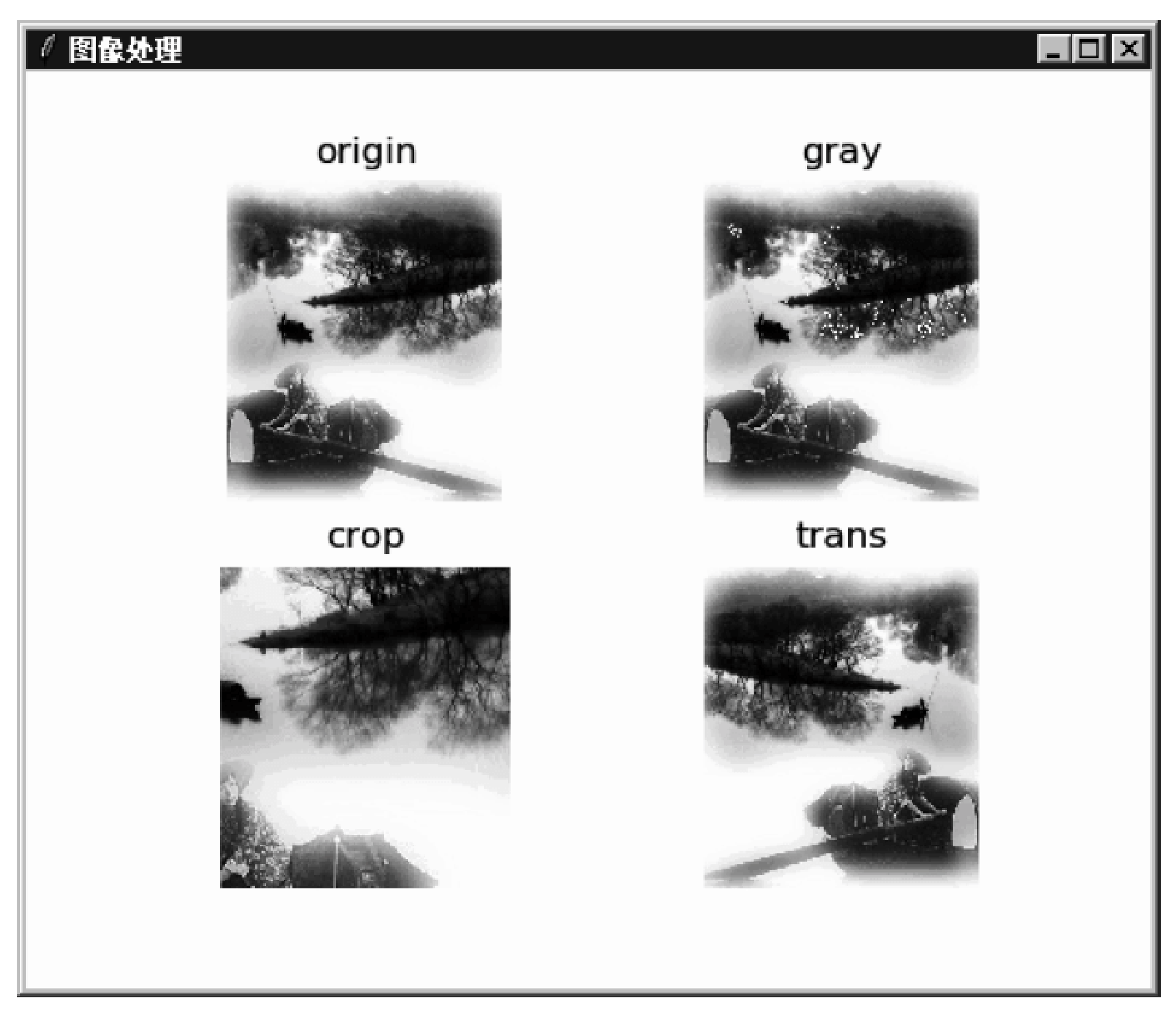
btn_show = Button(win, text='彩色转灰度', command=com_gray)
btn_show.grid(row=2, column=2)

btn_show = Button(win, text='裁剪图片', command=com_roi)
btn_show.grid(row=2, column=3)

btn_show = Button(win, text='图片水平翻转', command=com_trans)
btn_show.grid(row=2, column=4)

win.mainloop()
```

程序运行结果如图 5.13 所示。



（a）单击按钮，显示图像处理结果



（b）程序运行窗体

图 5.13　简易图像处理器

【例 5-13】 应用 matplotlib 模块绘制指数曲线。

（1）首先用 arange()函数生成一个等差数列的数组。

arange()函数的一般格式为：

```
arange(初值, 终值, 等差值)
```

例如，函数 arange(0，10，2)所创建的等差数列为[2, 4, 6, 8]。

（2）应用 matplotlib 模块 plt 类的 plot()方法绘制指数曲线图。

程序代码如下：

```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0, 5, 0.2)  # 生成等差数列，其中等差值为0.2
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')  # 绘制指数曲线
plt.show()
```

程序运行结果如图 5.14 所示。

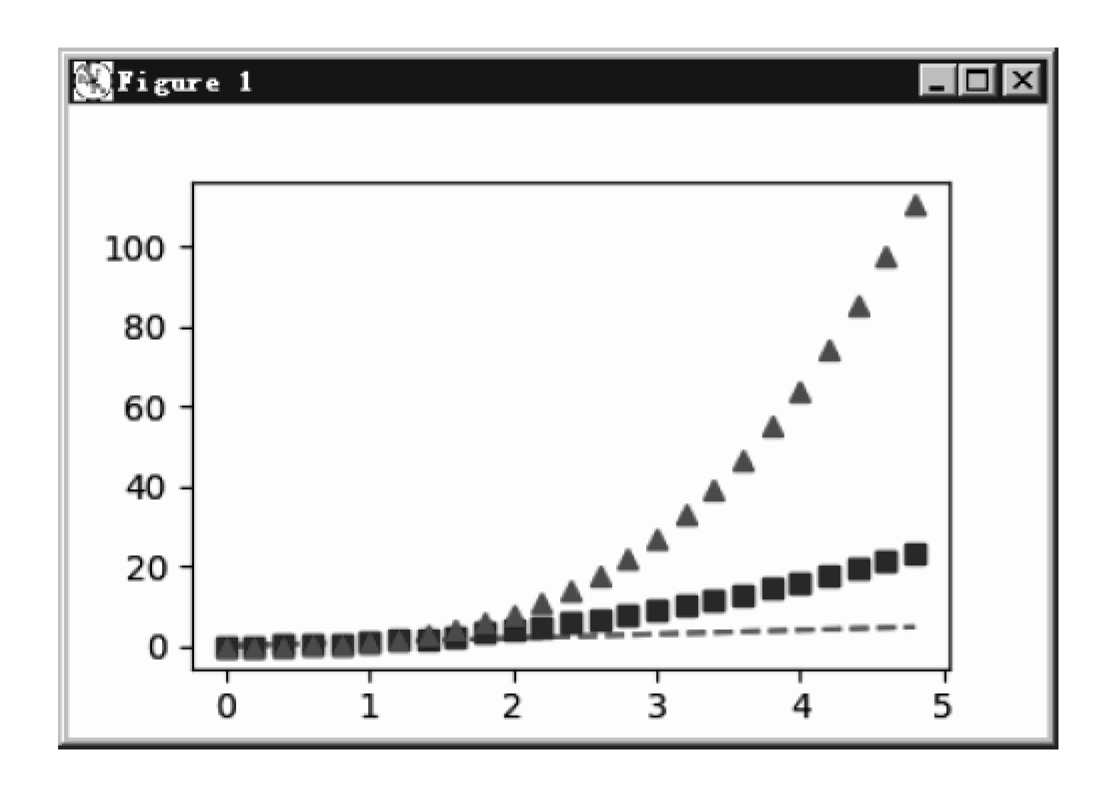


图 5.14　绘制曲线图形

习　题　5

1．绘制一个带阴影的小矩形块。

2．设计一个图片浏览器，单击“上一张”按钮，则显示前一张图片，单击“下一张”按钮，则显示后一张图片。

第 6 章　文件与数据库编程（数据存储）


视频录像

6.1　文 件 目 录

文件目录简称为目录，又称为文件夹，是文件系统中用于组织和管理文件的一种结构对象。对文件目录的主要操作有创建目录、删除目录、获取目录等。

Python 中对文件和目录的操作需要使用到 os 模块和 shutil 模块。

6.1.1　文件目录函数

Python 对文件目录操作定义了许多函数，常用的目录操作函数如表 6.1 所示。

表 6.1　常用的目录操作函数

函数	说明
os.mkdir("path")	创建目录
os.mkdirs("path")	创建多层目录
os.rmdir("dir")	只能删除空目录
shutil.rmtree("dir")	空目录、有内容的目录都可以删
os.rename("oldname","newname")	重命名目录
os.path.exists("path")	判断目录是否存在
os.path.isdir("path")	判断目标是否目录
shutil.copytree("olddir","newdir")	复制目录
shutil.move("olddir","newdir")	移动目录

6.1.2　文件目录操作

1. 创建文件目录

在 Python 中，应用 os 模块的 mkdir()函数创建文件目录，其语句格式如下：

```
os.mkdir(path)
```

其中，参数 path 为要创建的文件目录名。

【例 6-1】 创建一个名为 d:\py_test 的目录。

程序代码如下：

```
import os
os.mkdir("d:\\py_test")
```

将程序保存为 ex6_5.py，运行程序，则在 D 盘根目录下新建 py_test 目录。

【例 6-2】 创建一个名为 d:\mqtt\web 的多层文件目录。

程序代码如下：

```
# 导入os模块
import os
def mkdir(path):
    path=path.strip()             # 去除首位空格
    path=path.rstrip("\\")        # 去除尾部 \ 符号
    # 判断目录路径是否存在
    # 存在     True
    # 不存在   False
    isExists=os.path.exists(path)
    # 判断结果
    if not isExists:
        # 如果不存在则创建目录
        # 创建目录操作函数
        os.makedirs(path)
        print(path+' 创建成功')
        return True
    else:
        # 如果目录存在则不创建，并提示目录已存在
        print(path+' 目录已存在')
        return False

# 定义要创建的目录
mkpath= "d:\\mqtt\\web\\"
# 调用函数
mkdir(mkpath)
```

在本程序中，语句 os.path.exists(path)用来判断一个目录是否存在。

将程序保存为 ex6_2.py，运行程序，则在 D 盘根目录下新建两层文件目录 d:\mqtt\web。

2. 删除文件目录

删除文件目录的语句有两种：

（1）os.rmdir(path)

该语句只能删除空目录。

（2）shutil.rmtree(path)

该语句对于空目录或有内容的目录都可以删除。

【例 6-3】 删除例 6-2 所建立的 d:\mqtt 文件目录。

程序代码如下：

```
import shutil
import os

rmpath = "d:\\qttc"
isExists = os.path.exists(rmpath)
if isExists:   # 判断要删除的目录是否存在，若存在，则执行删除操作
    shutil.rmtree(rmpath)
    print('删除目录 ' + rmpath + ' 成功')
else:
    print('要删除的目录不存在！ ')
```

将程序保存为 ex6_3.py，运行程序后，则在 D 盘根目录下的 mqtt 文件目录已删除，该目录下的 web 子目录也一并被删除。

3. 复制文件目录

在 Python 中应用 shutil 模块的 copytree()函数复制文件目录，其语句格式如下：

```
shutil.copytree(oldpath, newpath)
```

其中，参数 oldpath 和 newpath 为目录名，目录 oldpath 必须存在且目录 newpath 必须不存在。

【例 6-4】 复制文件目录 d:\pytest 到 e:\test。

程序代码如下：

```
import shutil
import os

oldpath = "d:\\pytest"
newpath = "e:\\test"
isExists=os.path.exists(oldpath)
if isExists:
    shutil.copytree(oldpath, newpath)
    print('文件夹' + oldpath + '复制到' + newpath + '成功')
else:
    print('要复制的文件夹不存在！ ')
```

将程序保存为 ex6_4.py，运行程序后，文件目录 d:\pytest 连同该目录下的所有文件全部复制到 e:\test 下。

6.2 文件的读写操作

视频录像

6.2.1 文件操作函数

Python 对文件操作定义了许多函数，常用的文件操作函数如表 6.2 所示。

表 6.2　常用的文件操作函数

函数	说明
os.mknod("test.txt")	创建空文件
open("test.txt",w)	打开一个文件，如果文件不存在则创建文件
shutil.copyfile("oldfile","newfile")	复制文件
os.rename("oldname","newname")	重命名文件
shutil.move("oldpos","newpos")	移动文件
os.remove("file")	删除文件
os.path.isfile("goal")	判断目标是否为文件
os.path.exists("goal")	判断文件是否存在

6.2.2　打开和关闭文件

1. 打开文件

在 Python 中，使用 open 函数，可以打开一个已经存在的文件，或创建一个新文件。打开文件时将创建一个文件对象。其一般格式为：

```
f = open(文件名，访问模式)
```

其中，f 为创建的文件对象，参数“访问模式”如表 6.3 所示。

表 6.3　文件参数“访问模式”

访问模式	处理方式	功能说明	
		文件存在时	文件不存在时
r	只读	以只读方式打开文本文件	返回 NULL
w	只写	以只写方式打开或创建文本文件，并将源文件内容清空	创建新文件
a	追加	以追加方式打开文本文件，允许在文件末尾写入数据	创建新文件
rb	只读	以只读方式打开二进制文件	返回 NULL
wb	只写	以只写方式打开或创建二进制文件，源文件内容清空	创建新文件
ab	追加	以追加方式打开二进制文件，允许在文件末尾写数据	创建新文件
r+	读写	以读写方式打开文本文件	返回 NULL
w+	读写	以读写方式打开或创建文本文件，源文件内容清空	创建新文件
a+	读写	以读写方式打开文本文件，允许读或在文件末尾追加数据	创建新文件
rb+	读写	以读写方式打开二进制文件	返回 NULL
wb+	读写	以读写方式打开或创建二进制文件，源文件内容清空	创建新文件
ab+	读写	以读写方式打开二进制文件，允许读或在文件末尾追加数据	

2. 关闭文件

文件操作完成之后，需要将文件对象关闭，其一般格式为：

```
f.close()
```

6.2.3　读取文件操作

Python 使用 read()函数、readline()函数、readlines()函数实现读取文件的操作。

1. read()函数

使用 read()函数可以读取文件内容，其一般格式为：

```
str = f.read([b])
```

其中：

- f 为文件对象；
- 参数 b 为指定读取的字节数，如果不指定，则读取全部内容；
- str 为字符串，存放读取的内容。

【例 6-5】 设有文件 a.txt，其文件内容为 Hello Python，编写程序读取该文件中的内容，并显示到屏幕上。

程序代码如下：

```
import os
f = open("a.txt", "r")
str = f.read()
print(str)
f.close()
```

将程序保存为 ex6_5.py，运行程序结果如下：

```
Hello Python
```

2. readline()函数

使用 readline()函数可以逐行读取文件的内容，其一般形式如下：

```
str = f.readline()
```

【例 6-6】 有文件“荷塘月色.txt”，其文件内容为：

```
荷塘月色
剪一段时光缓缓流淌，
流进了月色中微微荡漾，
弹一首小荷淡淡的香，
美丽的琴音就落在我身旁。
```

编写程序，用 readline()函数逐行读取文件的内容。

程序代码如下：

```
import os
f = open("荷塘月色.txt", "r")
while True:
    str = f.readline()
    print(str)
    if not str:
        break
f.close()
```

将程序保存为 ex6_6.py，运行程序结果如下：

荷塘月色
剪一段时光缓缓流淌，
流进了月色中微微荡漾，
弹一首小荷淡淡的香，
美丽的琴音就落在我身旁。

3. readlines()函数

使用 readlines()函数可以一次读取文件中所有行的内容，其一般形式如下：

```
str = f.readlines()
```

【例 6-7】 编写程序，用 readlines()函数读取例 6-6 中“荷塘月色.txt”的文件内容。

程序代码如下：

```
import os
f = open("荷塘月色.txt", "r")
str = f.readlines()
print(str)
f.close()
```

将程序保存为 ex6_7.py。程序运行结果如下：

```
['荷塘月色\n', '剪一段时光缓缓流淌\n', '流进了月色中微微荡漾\n', '弹一首小荷淡淡的
香\n', '美丽的琴音就落在我身旁']
```

6.2.4 写入文件操作

Python 通过函数 write()向文件写入数据，其一般格式为：

```
f.write(content)
```

其中，f 为文件对象；参数 content 为写入文件的数据内容。

【例 6-8】 编写程序，新建文本文件 ex6_8.txt，并向其写入文本数据。

程序代码如下：

```
import os
str = "Hello Python \n向文件写入数据"
f = open("ex6_8.txt", "w")
f.write(str)
f.close()
```

将程序保存为 ex6_8.py，运行程序后，在当前目录下生成一个名为 ex6_8.txt 的文本文件，其内容为：

```
Hello Python
向文件写入数据
```

【例 6-9】 编写程序，在文件 ex6_8.txt 原数据内容之后，添加“我对学习 Python 很

痴迷！”。

当以 w 模式为参数调用 open()函数打开文件时，如果写入数据到文件中，新内容将覆盖文件中原有数据内容。若要在文件中追加数据，可以以 a 或 a+模式为参数调用 open()函数打开文件。

程序代码如下：

```
import os
f1 = open("ex6_8.txt", "a+")
f1.write("\n我对学习Python很痴迷！")
f1.close()
f2 = open("ex6_8.txt", "r")
str = f2.read()
print(str)
```

将程序保存为 ex6_9.py，运行程序，其结果如下：

```
Hello Python
向文件写入数据
我对学习Python很痴迷！
```

【例 6-10】 编写一个具有保存和读取文件功能的简易记事本程序。

程序代码如下：

```
import tkinter
import datetime
import time
import os
from tkinter import *

root = tkinter.Tk()
root.title('简易记事本')

## 保存按钮事件
def saveText():
  # 在内容上方加一行 显示保存文件的时间
  msgcontent = time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()) +\
 ' 保存数据如下:\n\n'
  str1 = text_msg.get('0.0', 'end')
  text_msg.delete('0.0', 'end')
  text_msg.insert('end', msgcontent, 'green')
  text_msg.insert('end', str1)
  f1 = open("book.dat", "a+")
  f1.write(str1)
  f1.close()

##  读取按钮事件
def readText():
    text_msg.delete('0.0', 'end')
    f2 = open("book.dat", "r")
```

```
    str2 = f2.read()
    text_msg.insert('end',str2)
    f2.close()

## 创建几个frame作为容器
frame_left_center  = tkinter.Frame(width=280, height=200, bg='white')
frame_save  = tkinter.Frame(width=140, height=40)
frame_read  = tkinter.Frame(width=140, height=40)

## 创建需要的几个元素
text_msg = tkinter.Text(frame_left_center);
str1 = StringVar()
button_save = tkinter.Button(frame_save, text='保存文件', command=saveText)
button_read = tkinter.Button(frame_read, text='读取文件', command=readText)

# 创建一个绿色的tag
text_msg.tag_config('green', foreground='#008B00')

# 使用grid设置各个容器位置
frame_left_center.grid(row=1, column=0, padx=2, pady=5)
frame_save.grid(row=2, column=0,sticky='W')
frame_read.grid(row=2, column=0,sticky='E')
#frame_left_top.grid_propagate(0)
frame_left_center.grid_propagate(0)

# 把元素填充进frame
text_msg.grid()
button_save.grid()
button_read.grid()

# 主事件循环
root.mainloop()
```

将程序保存为 ex6_10.py，运行程序。在文本框中输入文字内容，单击“保存文件”按钮后，把输入的字符数据保存到 book.dat 文件中，如图 6.1 所示。



图 6.1 简易记事本

6.2.5 二进制文件的读写

以 rb+或 wb+模式调用 open()函数打开文件，可以对二进制文件进行读写操作。

【例 6-11】 设有图片文件 img1.gif，将其数据读出，并写入到新建的 img2.gif 文件中。

设计思路：首先从图片文件 img1.gif 中读取数据，将数据存放到变量 byte 中，再将存放在变量中的数据写到文件 img2.gif 中。

程序代码如下：

```
import os
import tkinter
from tkinter import *

master = Tk()
master.title('复制图片')

# 按钮事件
def copyImage():
  f1 = open("img1.gif", "rb+")
  byte = f1.read()
  f1.close()
  f2 = open("img2.gif", "wb+")
  f2.write(byte)
  f2.close()
  photo2 = PhotoImage(file='img2.gif')
  label_2 = Label(image=photo2)
  label_2.image = photo2
  label_2.grid(row=0, column=1)

photo1 = PhotoImage(file='img1.gif')
label_1 = Label(image=photo1)
label_1.image = photo1
label_1.grid(row=0, column=0)

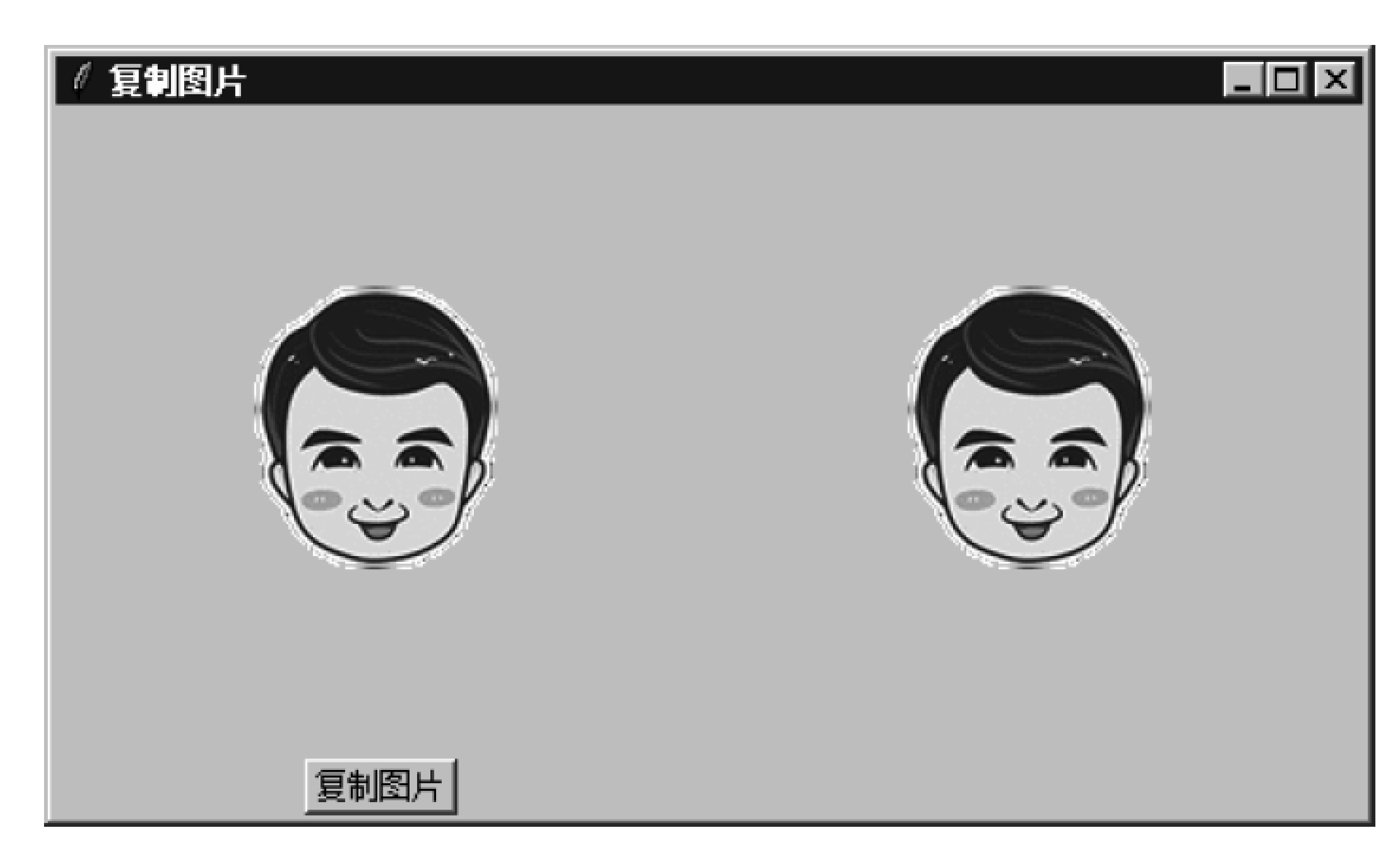
button_copy = tkinter.Button(master, text='复制图片', command=copyImage)
button_copy.grid(row=1,column=0)

master.mainloop()
```

将程序保存为 ex6_11.py，运行程序后，可以看到在当前目录中，新建了 img2.gif 文件，其图片内容与 img1.gif 完全一致，如图 6.2 所示。



（a）单击“复制图片”按钮之前



（b）单击“复制图片”按钮之后

图 6.2　复制图片

6.2.6　对 Excel 数据的读写操作

1. 安装 xlrd/xlwt 模块

Python 操作 Excel 电子表格数据需要用到 xlrd 模块和 xlwt 模块，xlrd 模块用于从 Excel 中读取数据；xlwt 模块用于向 Excel 中写入数据。

xlrd 模块和 xlwt 模块不是 Pyton 系统自带模块，因此在使用前必须用 pip 安装该模块。用 pip 安装 xlrd 模块的命令如下：

```
pip install xlrd
```

安装结果情况如图 6.3 所示。



图 6.3　安装 xlrd 模块

用 pip 安装 xlwt 模块的命令如下：

```
pip install xlwt
```

安装结果情况与安装 xlrd 模块相同，这里不再赘述。

2. 读取 Excel 表格中的数据

在 Python 中，读取 Excel 表格中的数据的主要步骤如下：

（1）导入 xlrd 模块

编写读取 Excel 表格数据的程序，首先需要导入 xlrd 模块：

```
import xlrd
```

（2）打开 Excel 文件读取数据，创建文件对象赋值给 workfile：

```
workbook = xlrd.open_workbook(r'd:\\pytest\\demo.xlsx')
```

（3）获取工作表，创建表格对象 table 有三种方法。

```
table = workbook.sheet_names()                # 获取所有工作表
table = workfile.sheet_by_index(0)            # 通过索引顺序获取
table = workfile.sheet_by_name('Sheet1')      # 通过名称获取
```

（4）获取行数和列数

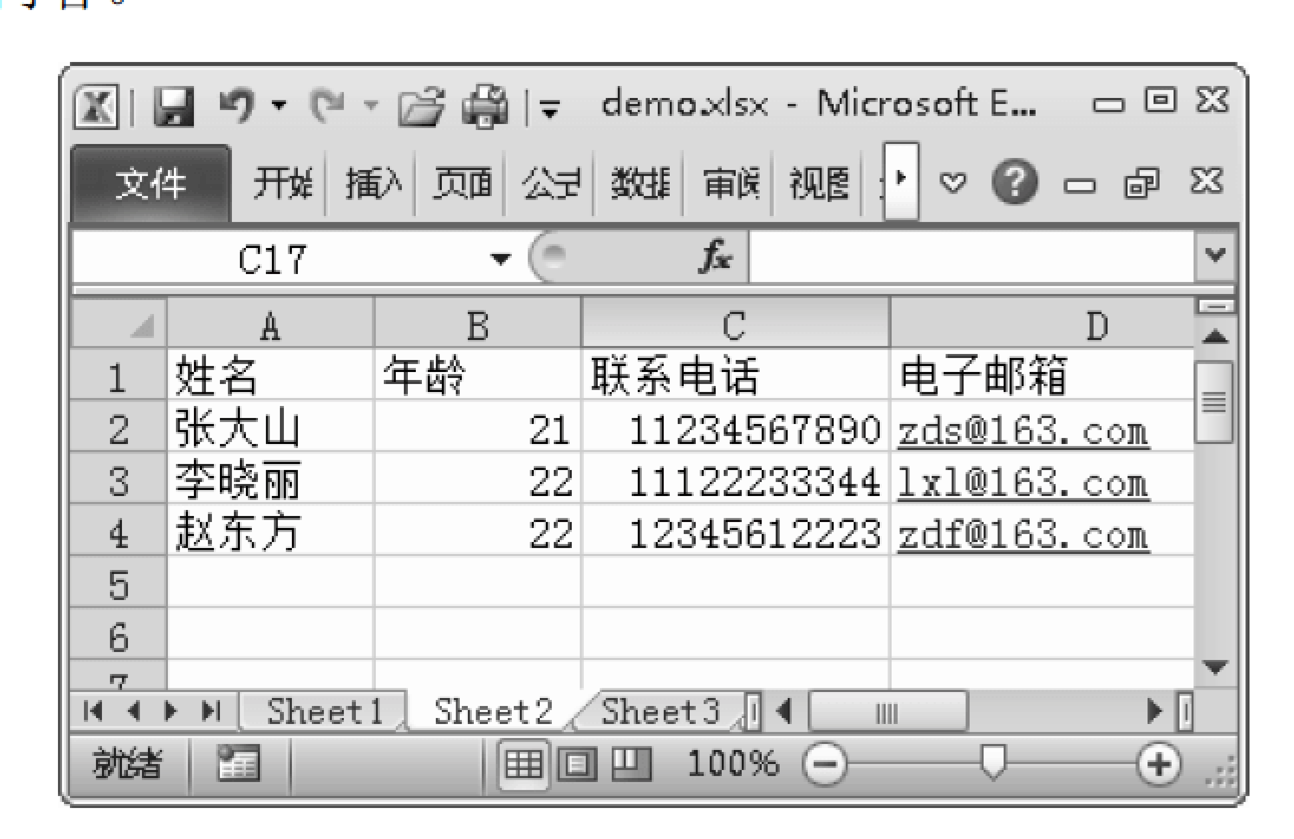
```
nrows = table.nrows                           # 获取工作表的行数
ncols = table.ncols                           # 获取工作表的列数
```

（5）获取指定单元格的数据，注意行和列的索引值都是从 0 开始

```
cell_A1 = table.cell(0,0).value               #表格中A1位置的数据
cell_C4 = table.cell(2,3).value               #表格中C4位置的数据
```

下面举例说明读取 Excel 表格的设计方法。

【例 6-12】 设有 Excel 表格 demo.xlsx，其中第 2 张电子表 Sheet2 的内容如图 6.4 所示。现读出其中的数据内容。



	A	B	C	D
1	姓名	年龄	联系电话	电子邮箱
2	张大山	21	11234567890	zds@163.com
3	李晓丽	22	11122233344	lxl@163.com
4	赵东方	22	12345612223	zdf@163.com

图 6.4　Excel 表格 Sheet2 的内容

程序代码如下：

```
# 导入xlrd模块
import xlrd

from datetime import date,datetime
```

```
def read_excel():
    # 打开文件
    workbook = xlrd.open_workbook(r'd:\\pytest\\demo.xlsx')
    # 获取所有sheet
    print("有数据表：")
    print(workbook.sheet_names())

     # 获取第2张表名
     sheet2_name = workbook.sheet_names()[1]

    # 根据sheet索引或名称获取sheet内容
    sheet2 = workbook.sheet_by_index(1)    # sheet索引从0开始
    sheet2 = workbook.sheet_by_name('Sheet2')

    # sheet的名称、行数和列数
    print(sheet2.name,sheet2.nrows,sheet2.ncols)

    # 获取整行和整列的值(数组元素从0开始计数)
    rows = sheet2.row_values(2)  # 获取第3行内容
    cols = sheet2.col_values(1)  # 获取第2列内容
    print(rows)
    print(cols)

    # 获取单元格内容
    print (sheet2.cell(1,0).value.encode('utf-8'))
    print (sheet2.cell_value(1,0).encode('utf-8'))
    print (sheet2.row(1)[0].value.encode('utf-8'))

    # 获取单元格内容的数据类型
    print (sheet2.cell(1,0).ctype)

if __name__ == '__main__':
    read_excel()
```

将程序保存为 ex6_12.py，运行程序结果如下：

```
有数据表:
['Sheet1', 'Sheet2', 'Sheet3']
Sheet2 4 4
['李晓丽', 22.0, 11122233344.0, 'lxl@163.com']
['年龄', 21.0, 22.0, 22.0]
b'\xe5\xbc\xa0\xe5\xa4\xa7\xe5\xb1\xb1'
b'\xe5\xbc\xa0\xe5\xa4\xa7\xe5\xb1\xb1'
b'\xe5\xbc\xa0\xe5\xa4\xa7\xe5\xb1\xb1'
1
```

3. 写入数据到 Excel 表格

写入数据到 Excel 表格的主要步骤如下：

（1）导入 xlwt 模块

```
import xlwt
```

（2）新建一个 Excel 文件

```
file = xlwt.Workbook()      # 注意Workbook首字母是大写
```

（3）新建一个 sheet 工作表

```
table = file.add_sheet('sheet name')
```

（4）写入数据 table.write(行,列,value)

```
table.write(0,0,'test')
```

（5）保存文件

```
file.save('Excel_test.xls')
```

【例 6-13】 新建 Excel 表格的示例。

程序代码如下：

```
import xlwt

wbk = xlwt.Workbook()
sheet = wbk.add_sheet('Mysheet1')
sheet.write(0,1,'test text')
sheet.write(1,1,'test text')
wbk.save('Excel_test.xls')
```

运行程序后，在当前目录下生成名为 Excel_test.xls 的 Excel 文件，如图 6.5 所示。

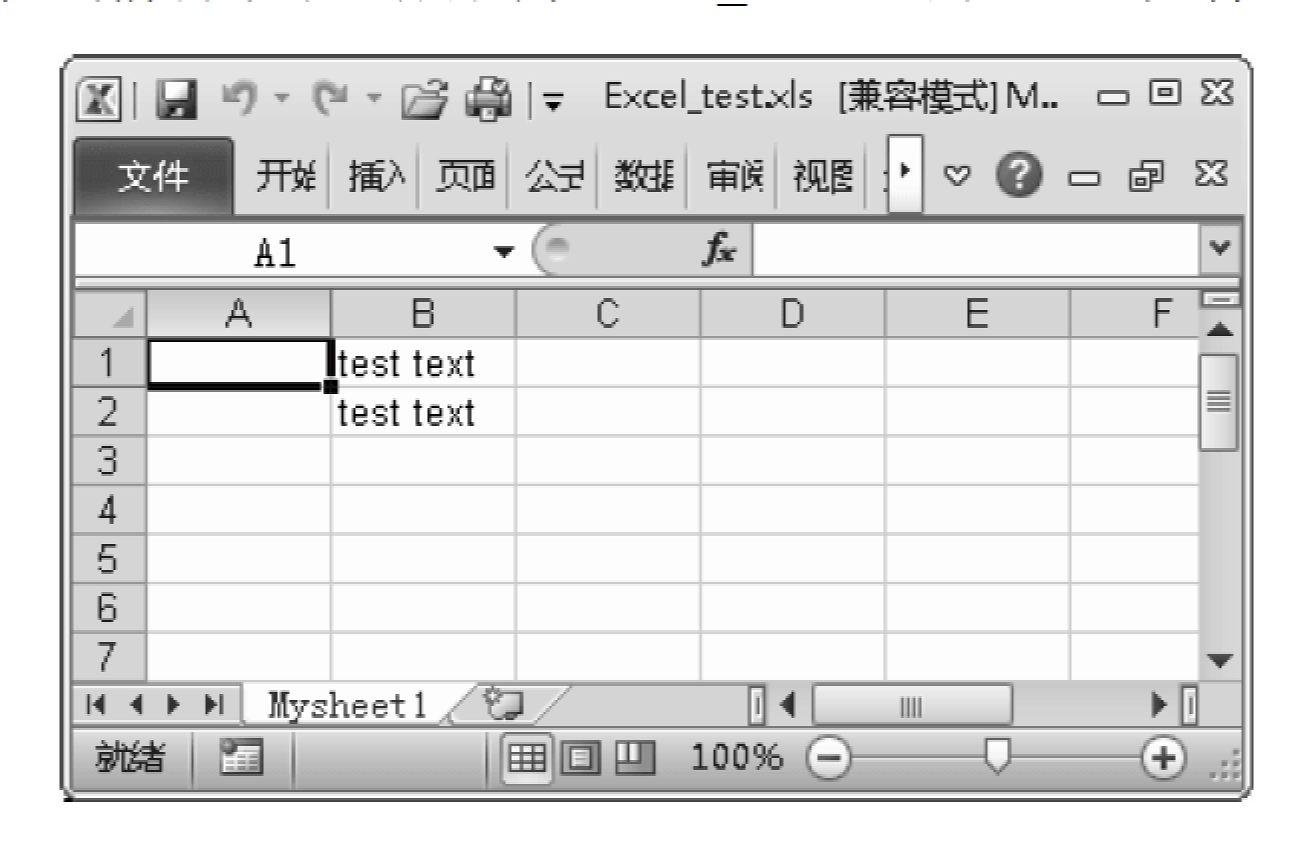


图 6.5　新生成的 Excel 文件

【例 6-14】 编写自定义风格样式的 Excel 表格。

程序代码如下：

```
import xlwt

workbook = xlwt.Workbook(encoding = 'ascii')
worksheet = workbook.add_sheet('My Worksheet')
style = xlwt.XFStyle()                               # 初始化样式
font = xlwt.Font()                                   # 为样式创建字体
font.name = 'Times New Roman'
font.bold = True                                     # 黑体
font.underline = True                                # 下画线
font.italic = True                                   # 斜体字
style.font = font                                    # 设定样式
worksheet.write(0, 0, 'Unformatted value')           # 不带样式的写入
worksheet.write(1, 0, 'Formatted value', style)# 带样式的写入
workbook.save('Excel_test2.xls')                     # 保存文件
```

运行程序后，在当前目录下生成名为 Excel_test2.xls 的 Excel 文件，如图 6.6 所示。

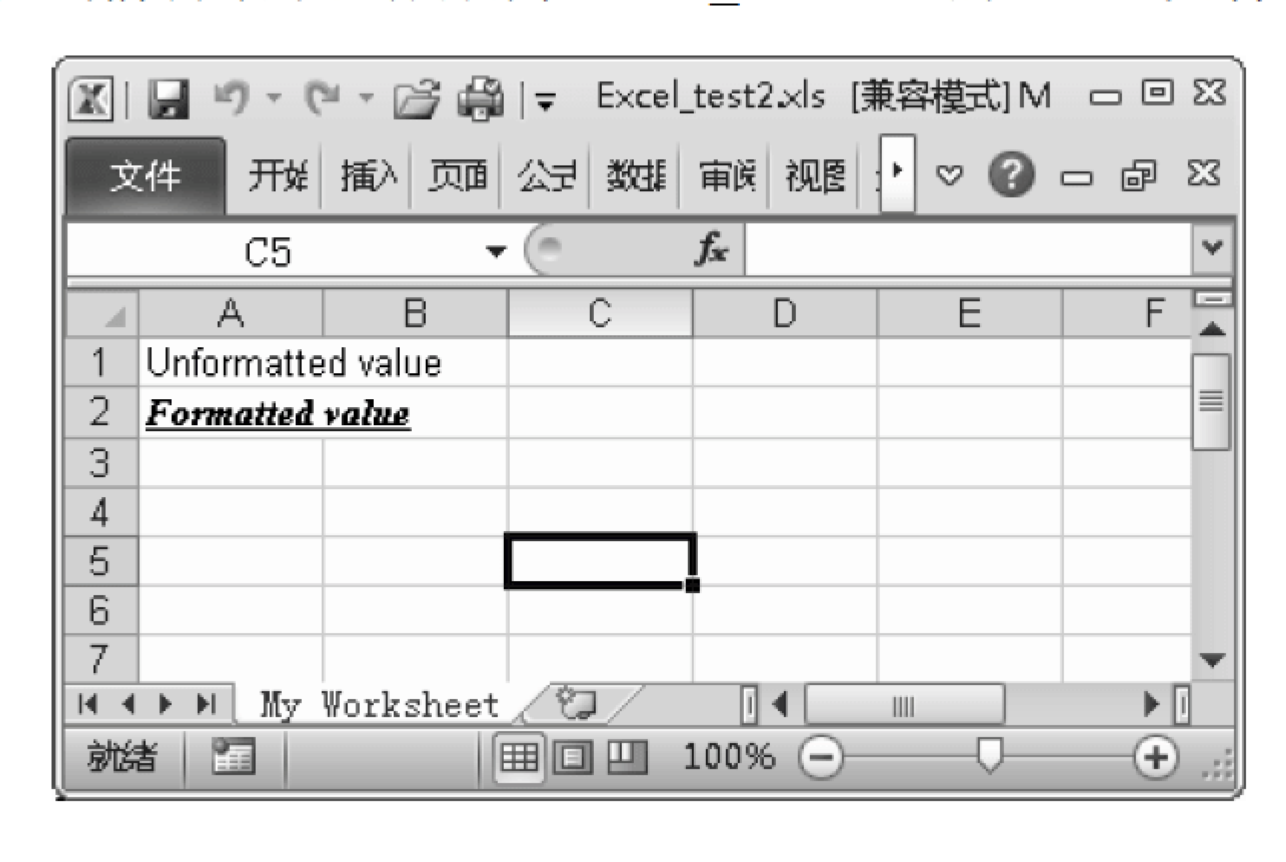


图 6.6　自定义风格样式的 Excel 文件

6.2.7　处理 JSON 格式数据

JSON（JavaScript Object Notation）是一种数据交换格式。JSON 采用完全独立于语言的纯文本格式，易于阅读和编写，同时也易于机器解析和生成（一般用于提升网络传输速率），因此 JSON 成为网络传输中理想的数据交换语言。

1. JSON 数据

JSON 数据可以是一个简单的字符串（String）、数值（Number）、布尔值（Boolean），也可以是一个数组或一个复杂的 Object 对象。

- JSON 的字符串需要用单引号或双引号括起来；
- JSON 的数值可以整数或浮点数；
- JSON 的布尔值为 True 或 False；
- JSON 的数组用方括号括起来；

- JSON 的 Object 对象用大括号括起来。

（1）用键-值对表示数据

JSON 数据的书写格式是：

```
键名(key) : 值(value)
```

键-值对的键名 key 必须是字符串，后面写一个冒号“:”，然后是值 value，值 value 可以是字符串、数值、布尔值。

例如：

```
'firstName' : 'John'
```

很容易理解，等价于下列赋值语句：

```
firstName = 'John'
```

（2） JSON 对象

JSON 对象可以包含多个键-值对，要求在大括号“{ }”中书写，键-值对之间用逗号“,”分隔。

例如：

```
{ "firstName":"John" , "lastName":"Doe" , "age":20 }
```

很容易理解，等价于下列 JavaScript 语句：

```
firstName = "John"
lastName = "Doe"
age = 20
```

JSON 对象的值也可以是另一个对象。例如：

```
{
    "Name":"John" ,
    "age": 20,
    "hobby":"打篮球",
    "friend":{"Name":"Suny" , "age":19 , "hobby":"看书"}
}
```

（3）JSON 数组

JSON 数组可以包含多个 JSON 数据作数组元素，每个元素之间用逗号“,”分隔，要求在方括号“[ ]”中书写。

例如：

```
meber = [ "John" , 20 , "打篮球" ]
```

JSON 数组的元素可以包含多个对象。例如：

```
employees = [
    {"sid":"a1001","name":"张大山","age": 21},
    {"sid":"a1002","name":"李晓丽","age": 20},
    {"sid":"a1003","name":"赵志坚","age": 22}]
```

访问 JavaScript 对象数组中的第一项元素语句如下：

```
employees[0].name;
```

返回的值为“张大山”。

也可以修改其数据：

```
employees[0].name = "张海山"
```

（4）JSON 文件

可以将 JSON 格式的数据保存为一个文件，该文件称为 JSON 文件，JSON 文件的文件类型是.json。

例如，将下列数据保存到文件 test.json 中：

```
{"name": " 百 度 ", "company_url": "http://www.baidu.com", "telephone":
"010-59928888", "crawl_time": "2017-06-13 16:11:16"}
```

2. JSON 模块

JSON 模块是由 Python 标准库提供的，该模块用一种很简单的方式对 JSON 数据进行解析，将 JSON 格式数据与 Python 标准数据类型相互转换。

常见的 Python 标准数据类型与 JSON 格式数据的转化对照如表 6.4 所示。

表 6.4　常见的 Python 数据类型与 JSON 格式数据的转化对照表

Python 数据类型	JSON 格式数据
dict	object
list	array
str	string
None	null

使用 JSON 模块时，需要使用导入模块语句：

```
import json
```

JSON 模块进行编码与解码的主要方法是 json.dumps()与 json.loads()和 json.dump()与 json.load()。

json.dumps(obj)方法将 JSON 对象 obj 类型转换成 Python 的数据类型，这个过程称为编码，json.loads(str)方法将 Python 数据类型转换回 JSON 对象数据类型，这个过程称为解码。

json.dump()方法把数据写入文件中，json.load()方法把文件中的数据读取出来。

3. 读取 JSON 数据

【例 6-15】 设有 JSON 格式数据：

```
data = {
    'name' : 'zhangdasan',
    'age' : 21,
    'email' : 'zdsan@163.com'
}
```

① 将数据编码转换成 Python 数据类型的数据。
② 将编码后的数据传回 JSON 对象，并输出对象各元素的值。
程序代码如下：

```
import json

data = {
    'name' : 'zhangdasan',
    'age' : 21,
    'email' : 'zdsan@163.com'
}

print('(1)编码为Python数据：')
json_str = json.dumps(data)          # 编码,将数据转换为字符串
print('Python数据：',json_str)
print('字符串长度：',len(json_str))

print('\n (2)解码为JSON对象:')
json_obj = json.loads(json_str)      # 解码,将字符串转换为JSON对象
j_name = json_obj['name']     ┐
j_age = json_obj['age']       ├ 获取 JSON 对象的元素
j_email = json_obj['email']   ┘
print(json_obj)
print('姓名：', j_name)
print('年龄：', j_age)
print('邮箱：', j_email)
```

程序运行结果如下：
① 编码为 Python 数据：

```
Python数据： {"name": "zhangdasan", "age": 21, "email": "zdsan@163.com"}
字符串长度： 59
```

② 解码为 JSON 对象：

```
{'name': 'zhangdasan', 'age': 21, 'email': 'zdsan@163.com'}
姓名： zhangdasan
年龄： 21
邮箱： zdsan@163.com
```

4. 读写 JSON 文件

使用 json.dump(obj)方法可以将数据写入 JSON 文件中；而 json.load(str)方法把文件打开后，以 JSON 对象的格式读取文件中的数据内容。

【例 6-16】 将一个 JSON 数据保存到 test1.json 文件中，然后从文件中读取数据，并显示到屏幕上。

程序代码如下：

```
import json

data = {
    'name' : 'zhangdasan',
    'age' : 21,
    'email' : 'zdsan@163.com'
}

f1 = open('test1.json', 'w')
json.dump(data, f1)
print('成功写入数据到文件！\n')
f1.close()

f2 = open('test1.json', encoding='utf-8')
line = f2.readline()
d = json.loads(line)
name = d['name']
age = d['age']
email = d['email']
print(name, age, email)
f2.close()
```

将数据写入 JSON 文件中

读取 JSON 文件数据

运行程序后，生成一个保存有 JSON 数据的文件 test1.json，并在屏幕上显示：

```
成功写入数据到文件！

zhangdasan  21  zdsan@163.com
```

【例 6-17】 将一批 JSON 数据保存到 test2.json 文件中，然后从文件中读取数据，并显示到屏幕上。

程序代码如下：

```
import json

data = [
{"sid":"a1001","name":"zhangdasan","age": 21},\
{"sid":"a1002","name":"lixianli","age": 20},\
{"sid":"a1003","name":"zhaozhijian","age": 22},\
]
```

```
f1 = open('test2.json', 'w')
json.dump(data, f1)
print('成功写入数据到文件！\n')
f1.close()

f2 = open('test2.json', encoding='utf-8')
line = f2.readline()
d = json.loads(line)
print(d)
for i in d:
    sid = i['sid']
    name = i['name']
    age = i['age']
    print(sid,name, age)
f2.close()
```

运行程序后，生成一个保存有 JSON 数据的文件 test2.json，并在屏幕上显示：

```
成功写入数据到文件！

[{'sid': 'a1001', 'name': 'zhangdasan', 'age': 21}, {'sid': 'a1002', 'name': 'li
xianli', 'age': 20}, {'sid': 'a1003', 'name': 'zhaozhijian', 'age': 22}]
a1001 zhangdasan 21
a1002 lixianli 20
a1003 zhaozhijian 22
```


视频录像

6.3 Python 数据库编程

Python 可以连接并使用各种数据库。下面分别以 SQLite 数据库和 MySQL 数据库为例，介绍编写 Python 程序对数据库进行操作的方法。

6.3.1 SQLite 数据库编程

SQLite 数据库是一个开源的嵌入式关系数据库，由于 Python 中集成了 SQLite 数据库模块，在 Python 程序中可以很方便地访问 SQLite 数据库。

1. SQLite 数据库简介

SQLite 数据库是一个关系型数据库，因为它很小，引擎本身只是一个大小不到 300KB 的文件，所以常作为嵌入式数据库内嵌在应用程序中。SQLite 生成的数据库文件是一个普通的文件，可以放置在任何目录下。SQLite 是用 C 语言开发的，开放源代码，支持跨平台，最大支持 2048GB 数据，并且被所有的主流编程语言支持。可以说，SQLite 是一个非常优秀的嵌入式数据库。

SQLite 数据库的管理工具很多，比较常用的有 SQLite Expert Professional，其功能强大，

几乎可以在可视化的环境下完成所有数据库操作。可以方便地使用它进行创建数据表和对数据记录进行增加、删除、修改、查询的操作。SQLite Expert Professional 的运行界面如图 6.7 所示。

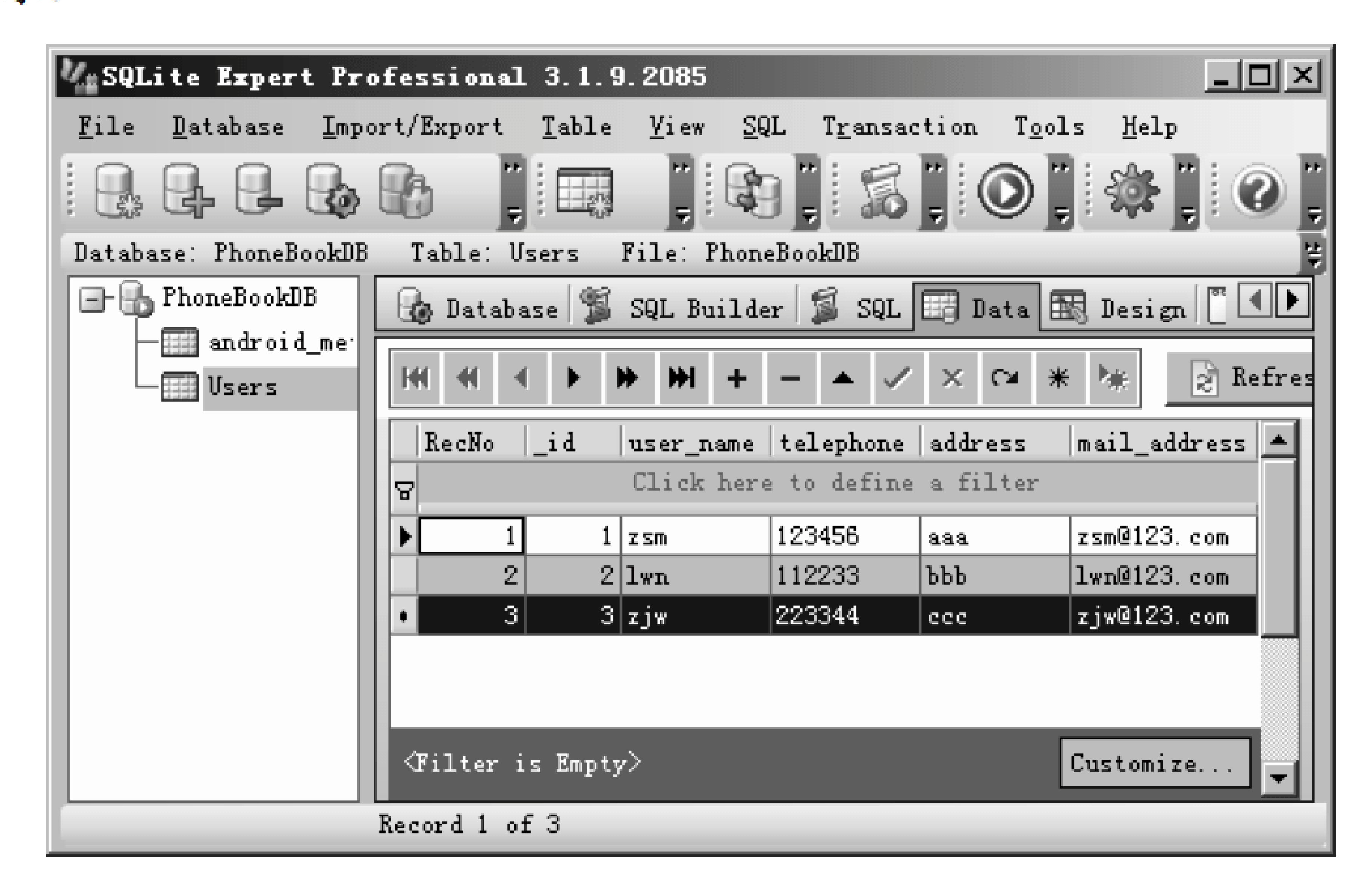


图 6.7　SQLite Expert Professional 的运行界面

在 Python 系统的内部嵌入了 SQLite 数据库模块 sqlite3，所以 Python 应用程序可以很方便地使用 SQLite 数据库存储数据。

2. sqlite3 模块

sqlite3 模块是 Python 操作 SQLite 数据库的接口模块。在 sqlite3 模块中定义了一系列连接和操作数据库的方法，其常用方法如表 6.5 所示。

表 6.5　sqlite3 模块的常用方法

方法	说明
sqlite3.connect(database [,timeout ,other optional arguments])	连接到一个 SQLite 数据库文件，如果连接的数据库不存在，则创建一个数据库文件
connection.cursor([cursorClass])	创建一个游标 cursor
cursor.execute(sql [, optional parameters])	执行 SQL 命令的语句
connection.execute(sql [, optional parameters])	执行 SQL 命令的语句
connection.commit()	提交事务
connection.close()	关闭数据库连接

3. 操作 SQLite 数据库

（1）连接数据库

应用 sqlite3 模块的 connect()方法可以连接 SQLite 数据库文件，如果数据库文件不存在，则创建一个 SQLite 数据库文件。

【例 6-18】 创建一个名为 test.db 的 SQLite 数据库文件。

程序代码如下：

```
import sqlite3
```

```
conn = sqlite3.connect("D:/test.db")
print("connection database successfully")
```

运行程序，可以看到，在 D:\目录下创建了一个 test.db 文件，这个文件就是 SQLite 数据库文件。

（2）创建数据表

在数据库 test.db 中，创建一个名为 user 的数据表，其表的结构如表 6.6 所示。

表 6.6　数据表 user 的结构

字段	类型	长度	注释
sid	INT	5	编号
name	VARCHAR	10	姓名
email	VARCHAR	25	邮箱

【例 6-19】 在数据库 test.db 中创建 user 数据表。

程序代码如下：

```
import sqlite3
conn = sqlite3.connect("d:/test.db")
sqlstr = "create table user (sid varchar(5) primary key, \
name varchar(10), email varchar(25))"
conn.execute(sqlstr)
print("create table successfully")
conn.close()
```

（3）添加数据记录

在 user 数据表中添加数据记录如下：

```
1001  张大山   zhangds@163.com
1002  李晓丽   lixli@163.com
1003  赵四方   zaoshi@163.com
```

【例 6-20】 在 user 数据表中，添加数据记录。

程序代码如下：

```
import sqlite3
conn = sqlite3.connect("d:/test.db")
cur = conn.cursor()

sqlstr1 = "insert into user(sid, name, email)\
values(1001, '张大山', 'zhangds@163.com') "
cur.execute(sqlstr1)

sqlstr2 = "insert into user(sid, name, email)\
values(1002, '李晓丽', 'lixli@163.com') "
cur.execute(sqlstr2)
```

```
sqlstr3 = "insert into user(sid, name, email)\
values(1003, '赵四方', 'zaoshi@163.com') "
cur.execute(sqlstr3)
conn.commit()

print("Records created successfully")
conn.close()
```

（4）查询记录

【例 6-21】 应用 SQL 命令的 select 查询语句显示记录。

程序代码如下：

```
import sqlite3

conn = sqlite3.connect("d:/test.db")
cur = conn.cursor()

sqlstr = "select * from  user"
s = cur.execute(sqlstr)
for row in s:
    print("sid=",row[0])
    print("name=",row[1])
    print("email=",row[2],'\n')

conn.close()
```

程序运行结果如下：

```
sid= 1001
name= 张大山
email= zhangds@163.com

sid= 1002
name= 李晓丽
email= lixli@163.com

sid= 1003
name= 赵四方
email= zaoshi@163.com
```

（5）修改数据记录

【例 6-22】 应用 SQL 命令的 update 语句修改数据记录。

程序代码如下：

```
import sqlite3
```

```
conn = sqlite3.connect("d:/test.db")
cur = conn.cursor()

sql_update = "update user set email='zhaosf@abc.com' where sid=1003"
cur.execute(sql_update)
conn.commit()

sql_select = "select * from  user where sid=1003"
s = cur.execute(sql_select)
for row in s:
    print("sid=",row[0])
    print("name=",row[1])
    print("email=",row[2],'\n')

conn.close()
```

程序运行结果如下：

```
sid= 1003
name= 赵四方
email= zhaosf@abc.com
```

（6）删除数据记录

【例 6-23】 应用 SQL 命令的 delete 语句删除 user 表中的第二条记录。

程序代码如下：

```
import sqlite3

conn = sqlite3.connect("d:/test.db")
cur = conn.cursor()

sql_update = "delete from user  where sid=1002"
cur.execute(sql_update)
conn.commit()

sql_select = "select * from  user"
s = cur.execute(sql_select)
for row in s:
    print("sid=",row[0])
    print("name=",row[1])
    print("email=",row[2],'\n')

conn.close()
```

程序运行结果如下：

```
sid= 1001
name= 张大山
email= zhangds@163.com

sid= 1003
name= 赵四方
email= zhaosf@abc.com
```

6.3.2 操作 MySQL 数据库

1. 安装 pymysql 模块

Python 需要引用 pymysql 模块来进行 MySQL 数据库的操作。

（1）用 pip 安装 pymysql 模块

在控制台命令窗口中，输入命令：

```
pip  install  pymysql
```

则包管理工具 pip 会自动完成 pymysql 模块的安装，如图 6.8 所示。

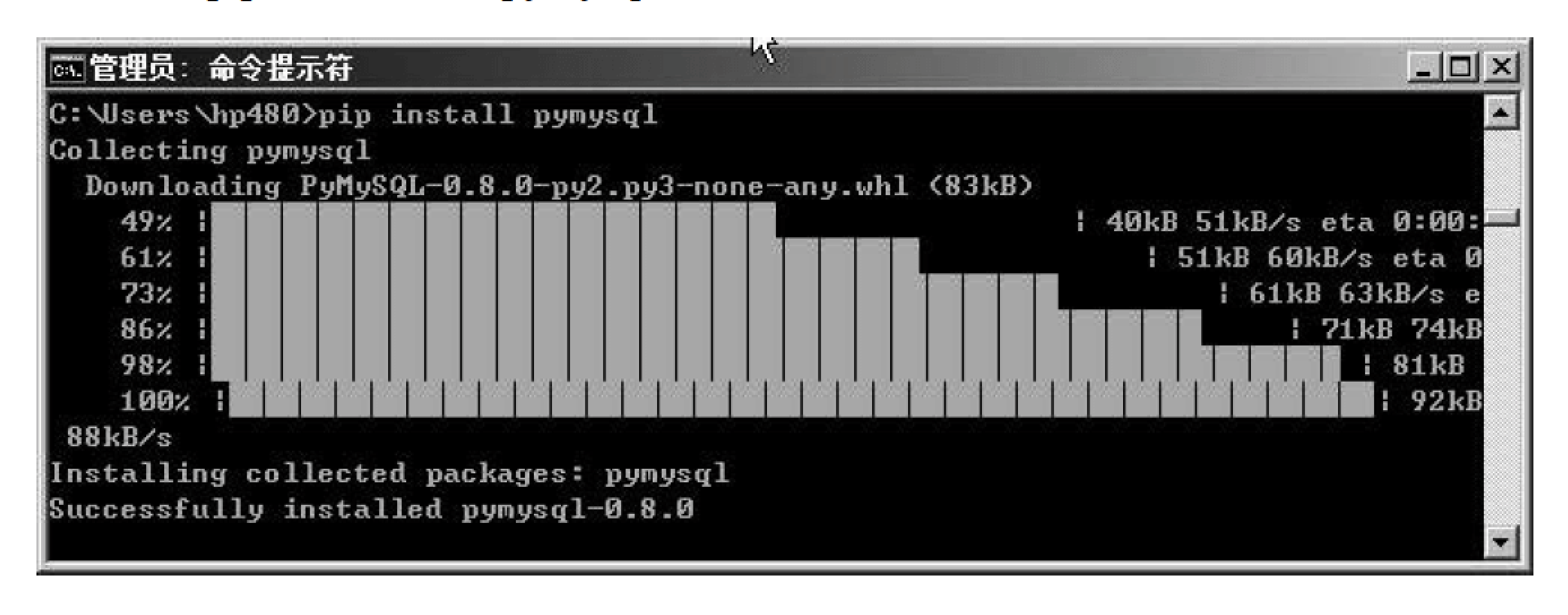


图 6.8　pip 安装 pymysql 模块

（2）测试 pymysql 模块是否安装成功

编写一个 Python 文件，输入导入 pymysql 的语句：

```
import pymysql
```

如果编译未出错，即表示 pymysql 安装成功。

2. pymysql 模块的方法

pymysql 模块定义了连接数据库及数据库连接对象的方法，其常用方法如下。

（1）pymysql.Connect()方法

pymysql.Connect()方法用于连接数据库并创建连接对象。其一般格式为

```
connection = pymysql.Connect(  host(str),     #  MySQL服务器地址
                               port(int),     #  MySQL服务器端口号
                               user(str),     #  用户名
                               passwd(str),   #  密码
```

```
                    db(str),        #  数据库名称
                    charset(str),   #  连接编码
                )
```

（2）数据库连接对象 connection 支持的方法

创建数据库连接后，其返回值为数据库连接对象。连接对象用于对数据库进行各种操作，其主要方法如表 6.7 所示。

表 6.7　数据库连接对象的主要方法

方法	说明
cursor()	创建并返回游标对象
commit()	提交当前事务
rollback()	回滚当前事务
close()	关闭连接

（3）游标对象支持的方法

由数据库连接对象所创建的游标对象主要用于对数据库的记录集进行各种操作，其主要方法如表 6.8 所示。

表 6.8　数据库连接对象所创建的游标对象的主要方法

方法	说明
execute(op)	执行一个数据库的查询命令
fetchone()	取得结果集的下一行
fetchmany(size)	获取结果集的下几行
fetchall()	获取结果集中的所有行
rowcount()	返回数据条数或影响行数
close()	关闭游标对象

3. Python 对 MySQL 数据库的操作

（1）创建数据库连接对象

使用 connect()方法可以创建数据库连接对象和打开数据库的连接，其方法如下：

```
数据库连接对象 = pymysql.connect(数据库服务器,用户名,密码,数据库名)
```

connect()方法返回一个数据库连接对象，通过数据库连接对象可以对数据库进行各种操作。

（2）创建游标对象

在 Python 中，使用游标可以执行 SQL 语句和查询数据。创建一个游标对象的方法如下：

```
游标对象 = 数据库连接对象.cursor()
```

【例 6-24】 假设 MySQL 数据库服务器中有一个名为 testdb 的数据库，并且数据库中有一个名为 trade 的数据表，其数据表的结构如表 6.9 所示。

表 6.9　数据表 trade 的结构

字段	类型	长度	说明
sid	INT	4	编号
name	VARCHAR	10	用户姓名
account	VARCHAR	11	银行储蓄账号
saving			账户储蓄金额
expend			账户支出总计
income			账户收入总计

（1）创建数据表

在连接数据库之前，需要创建一个数据表，方便测试 pymysql 的功能。创建数据表的语句如下：

```
DROP TABLE IF EXISTS 'trade';

CREATE TABLE 'trade' (
  'id' int(4) unsigned NOT NULL AUTO_INCREMENT,
  'name' varchar(6) NOT NULL COMMENT '用户真实姓名',
  'account' varchar(11) NOT NULL COMMENT '银行储蓄账号',
  'saving' decimal(8,2) unsigned NOT NULL DEFAULT '0.00' COMMENT '账户储蓄金额',
  'expend' decimal(8,2) unsigned NOT NULL DEFAULT '0.00' COMMENT '账户支出总计',
  'income' decimal(8,2) unsigned NOT NULL DEFAULT '0.00' COMMENT '账户收入总计',
  PRIMARY KEY ('id'),
  UNIQUE KEY 'name_UNIQUE' ('name')
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
INSERT INTO 'trade' VALUES (1,'张大山','18012345678',0.00,0.00,0.00);
```

可以将上面内容保存为数据库备份文件 trade.sql，再将其导入到 MySQL 的名为 testdb 的数据库中。

（2）编写程序，实现增、删、改、查功能和事务处理

程序代码如下：

```
import pymysql
import pymysql.cursors

# 连接数据库
connect = pymysql.Connect(
    host='localhost',
    port=3310,
    user='woider',
    passwd='3243',
    db='python',
    charset='utf8'
)
```

```
# 获取游标
cursor = connect.cursor()

# 插入数据
sql = "INSERT INTO trade (id.name, account, saving) VALUES ('%d', '%s', '%s', %.2f )"
data = (2,'李晓丽', '13512345678', 10000)
cursor.execute(sql % data)
connect.commit()
print('成功插入', cursor.rowcount, '条数据')

# 修改数据
sql = "UPDATE trade SET saving = %.2f WHERE account = '%s' "
data = (8888, '13512345678')
cursor.execute(sql % data)
connect.commit()
print('成功修改', cursor.rowcount, '条数据')

# 查询数据
sql = "SELECT name,saving FROM trade WHERE account = '%s' "
data = ('13512345678',)
cursor.execute(sql % data)
for row in cursor.fetchall():
    print("Name:%s\tSaving:%.2f" % row)
print('共查找出', cursor.rowcount, '条数据')

# 删除数据
sql = "DELETE FROM trade WHERE account = '%s' LIMIT %d"
data = ('13512345678', 1)
cursor.execute(sql % data)
connect.commit()
print('成功删除', cursor.rowcount, '条数据')

# 事务处理
sql_1 = "UPDATE trade SET saving = saving+1000 WHERE account = '18012345678' "
sql_2 = "UPDATE trade SET expend = expend+1000 WHERE account = '18012345678' "
sql_3 = "UPDATE trade SET income = income+2000 WHERE account = '18012345678' "

try:
    cursor.execute(sql_1)    # 储蓄增加1000
    cursor.execute(sql_2)    # 支出增加1000
    cursor.execute(sql_3)    # 收入增加2000
except Exception as e:
    connect.rollback()       # 事务回滚
    print('事务处理失败', e)
else:
```

```
    connect.commit()          # 事务提交
    print('事务处理成功', cursor.rowcount)

# 关闭连接
cursor.close()
connect.close()
```

程序运行结果如下：

```
成功插入 1 条数据
成功修改 1 条数据
Name:李晓丽 Saving: 8888.00
共查找出 1 条数据
成功删除 1 条数据
事务处理成功 1
```



视频录像

6.4 案例精选

6.4.1 多功能文本编辑器

【例 6-25】 编写一个多功能的文本编辑器。

程序代码如下：

```
from tkinter import *
import tkinter
import tkinter.filedialog
import tkinter.colorchooser
import tkinter.messagebox
import tkinter.scrolledtext


#from Tkinter import *
#from tkMessageBox import *
#from tkFileDialog import *
import os

#创建窗体
root = Tk()
root.title('多功能文本编辑器')
root.geometry("800x500+100+100")

#定义文件名变量
filename = ''
```

```
def author():
    tkinter.messagebox.showinfo('作者','sundy')

def about():
    tkinter.messagebox.showinfo('关于','多功能文本编辑器\n(v1.0版)')

def openfile():
    global filename
    filename = tkinter.filedialog.askopenfilename(
    title='打开文件', filetypes = [('Text files', '*.txt')])
    if filename == '':
        filename = None
    else:
        root.title('FileName:'+os.path.basename(filename))
        textPad.delete(1.0,END)
        f = open(filename,'r')
        textPad.insert(1.0,f.read())
        f.close()

def new():
    global filename
    root.title('未命名文件')
    filename = None
    textPad.delete(1.0,END)

def save():
    global filename
    try:
        f = open(filename,'w')
        msg = textPad.get(1.0,END)
        f.write(msg)
        f.close()
    except:
        saveas()

def saveas():
f = tkinter.filedialog.asksaveasfilename(title='保存文件',\
      initialfile= '未命名.txt', filetypes = [('Text files', '*.txt')])
print(f)
if f != '':
    global filename
    filename = f
    fh = open(f,'w')
    msg = textPad.get(1.0,END)
    fh.write(msg)
```

```
        fh.close()
        root.title('FileName:'+os.path.basename(f))

def cut():
    textPad.event_generate('<<Cut>>')

def copy():
    textPad.event_generate('<<Copy>>')

def paste():
    textPad.event_generate('<<Paste>>')

def redo():
    textPad.event_generate('<<Redo>>')

def undo():
    textPad.event_generate('<<Undo>>')

def selectAll():
    textPad.tag_add('sel','1.0',END)

def search():
    def dosearch():
        myentry = entry1.get()     # 获取查找的内容--string型
        whatever = str(textPad.get(1.0,END))
        tkinter.filedialog.showinfo("查找结果: ",\
          "you searched %s, \
          there are %d in the text"%(myentry,whatever.count(myentry)))
    topsearch = Toplevel(root)
    topsearch.geometry('300x30+200+250')
    label1 = Label(topsearch,text='Find')
    label1.grid(row=0, column=0,padx=5)
    entry1 = Entry(topsearch,width=20)
    entry1.grid(row=0, column=1,padx=5)
    button1 = Button(topsearch,text='查找',command=dosearch)
    button1.grid(row=0, column=2)

#Create Menu
menubar = Menu(root)
root.config(menu = menubar)

filemenu = Menu(menubar)
filemenu.add_command(label='新建', accelerator='Ctrl + N', command= new)
filemenu.add_command(label='打开', accelerator='Ctrl + O',command = openfile)
filemenu.add_command(label='保存', accelerator='Ctrl + S', command=save)
```

```
filemenu.add_command(label='另存为', accelerator='Ctrl + Shift + S',\
command=saveas)
menubar.add_cascade(label='文件',menu=filemenu)

editmenu = Menu(menubar)
editmenu.add_command(label='撤销', accelerator='Ctrl + Z', command=undo)
editmenu.add_command(label='重做', accelerator='Ctrl + y', command=redo)
editmenu.add_separator()
editmenu.add_command(label = "剪切",accelerator = "Ctrl + X",command=cut)
editmenu.add_command(label = "复制",accelerator = "Ctrl + C", command=copy)
editmenu.add_command(label = "粘贴",accelerator = "Ctrl + V", command= paste)
editmenu.add_separator()
editmenu.add_command(label = "查找",accelerator = "Ctrl + F", command=search)
editmenu.add_command(label = "全选",accelerator = "Ctrl + A", command= selectAll)

menubar.add_cascade(label = "操作",menu = editmenu)
aboutmenu = Menu(menubar)
aboutmenu.add_command(label = "作者", command=author)
aboutmenu.add_command(label = "关于", command = about)
menubar.add_cascade(label = "about",menu=aboutmenu)

#toolbar
toolbar = Frame(root, height=25,bg='grey')
shortButton = Button(toolbar, text='打开',command = openfile)
shortButton.pack(side=LEFT, padx=5, pady=5)
shortButton = Button(toolbar, text='保存', command = save)
shortButton.pack(side=LEFT, padx=5, pady=5)
shortButton = Button(toolbar, text='退出', command = exit)
shortButton.pack(side=LEFT, padx=5, pady=5)
toolbar.pack(expand=NO,fill=X)

#Status Bar
status = Label(root, text='Ln20',bd=1, relief=SUNKEN,anchor=W)
status.pack(side=BOTTOM, fill=X)

#linenumber&text
lnlabel = Label(root, width=2, bg='antique white')
lnlabel.pack(side=LEFT, fill=Y)
textPad = Text(root, undo=True)
textPad.pack(expand=YES, fill=BOTH)
scroll = Scrollbar(textPad)
textPad.config(yscrollcommand=scroll.set)
scroll.config(command = textPad.yview)
scroll.pack(side=RIGHT,fill=Y)

root.mainloop()
```

程序运行结果如图 6.9 所示。



图 6.9 多功能文本编辑器

6.4.2 保存结构化数据

1. 数据对象序列化后保存到文件

【例 6-26】 将字典结构的数据保存到文件中。

程序代码如下：

```
import pickle

data = [{'sid':'a1001','name':'张大山','scro':92},
        {'sid':'a1002','name':'李晓丽','scro':82},
        {'sid':'a1003','name':'赵志勇','scro':97}]

s_file = open('dic_data.dat', 'wb')
pickle.dump(data, s_file)
s_file.close()

r_file = open('dic_data.dat', 'rb')
data2 = pickle.load(r_file)
print('学号\t姓名\t成绩')
for i in data2:
    sid = i['sid']
    name = i['name']
    scro = i['scro']
    print(sid, '\t',name, '\t',scro)
r_file.close()
```

字典结构序列化后保存到文件

反序列化后读出数据

程序运行结果如下：

```
学号      姓名      成绩
a1001     张大山     92
```

```
a1002    李晓丽    82
a1003    赵志勇    97
```

2. 把任意对象保存到文件

【例 6-27】 在画布上绘制一个笑脸，将其保存到一个文件中。再从文件中读取“笑脸”对象，在画布中显示。

程序代码如下：

```
import tkinter
import tkinter.messagebox
import pickle

# 定义“笑脸”类
class smile:
    def create_can(self, can):
        self.can = can
        io1 = self.can.create_oval(35,30,210,210, fill='yellow')# 画一黄色圆
        io2 = self.can.create_oval(70,70,180,180, fill='black')
        io3 = self.can.create_oval(65,70,185,170, outline='yellow',\
                                    fill='yellow')
        io4 = self.can.create_oval(80,100,110,130, fill='black')
        io5 = self.can.create_oval(150,100,180,130, fill='black')

win = tkinter.Tk()
win.title('画布示例')
win.geometry('250×250')

can1 = tkinter.Canvas(win, height=220, width=220)      # 定义画布1
can1.grid(row=0, column=0, columnspan=2)

can2 = tkinter.Canvas(win, height=220, width=220)      # 定义画布2
can2.grid(row=0, column=2, columnspan=2)

def ccan():
    global can1                     }
    s_obj = smile()                 }  创建笑脸对象，绘制笑脸
    s_obj.create_can(can1)          }

# 将“笑脸”对象保存到文件中
def save_file():                         }
    s_file = open('smile.dat', 'wb')     }
    s_obj = smile()                      }  将笑脸对象序列化后，保存到文件中
    pickle.dump(s_obj, s_file)           }
    s_file.close()                       }
```

```
# 读取文件中的“笑脸”对象
def show_smile():
    r_file = open('smile.dat', 'rb')
    global can2
    obj = pickle.load(r_file)
    obj.create_can(can2)
```

从文件中读取笑脸对象，反序列化后显示

```
btn1=tkinter.Button(win, text='显示笑脸', command=ccan)
btn1.grid(row=1, column=0)

btn2=tkinter.Button(win, text='写入文件', command=save_file)
btn2.grid(row=1, column=1)

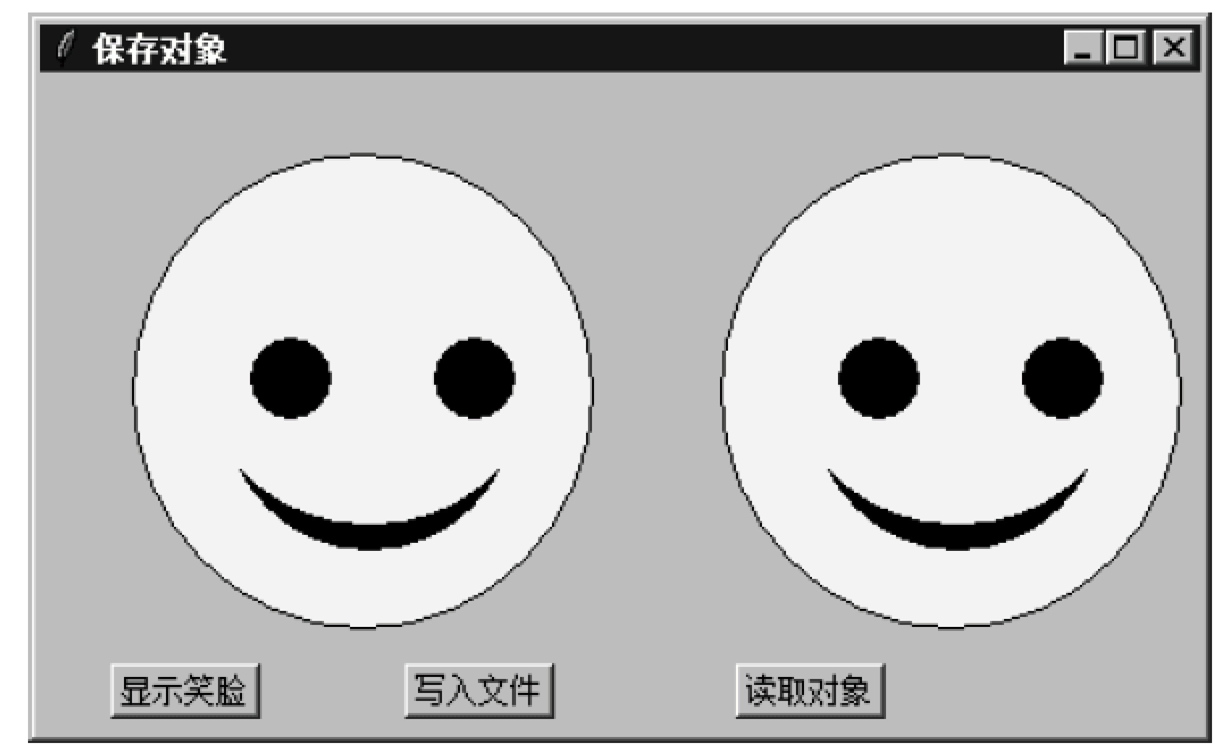
btn3=tkinter.Button(win, text='读取对象', command=show_smile)
btn3.grid(row=1, column=2)

win.mainloop()
```

运行程序，先单击“写入文件”按钮，把“笑脸”对象序列化后保存到文件 smile.dat 中；然后单击“读取对象” 按钮，可以看到，将保存在文件中的“笑脸”对象显示到画布上，如图 6.10 所示。



（a）显示笑脸



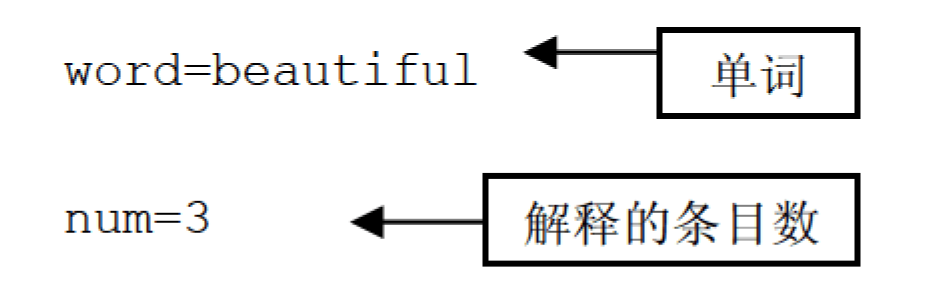
（b）读取文件中的“笑脸”对象

图 6.10 “笑脸”对象

6.4.3 英汉小词典设计

【例 6-28】 设计一个英汉小词典。

设有英汉解释文件 dic.txt，其格式为：



```
1
美丽的；漂亮的；
2
美好的；极好的；美妙的；
3
巧妙的；娴熟的；漂亮的；
```

解释

程序代码如下：

```
import os
#import sqlite3
from tkinter import *

class One_Word(object):
    def __init__(self):
        self.en = "u"
        self.num = 0
        self.chs = []
    def set_word(self, en, num, chs):
        self.num = num
        self.chs = chs
        self.en = en

def read_file():
    words = []
    with open('dic.txt','r',encoding='utf8') as f:
        while True:
          line = f.readline().strip('\n')
          if line == "":
              break
          wod = line.split("=")
          en = wod[1]
          nums = f.readline().strip('\n').split("=")
          num = int(nums[1])
          i=0
          chs = []
          while i<num:
              f.readline()
              chs.append(f.readline().strip('\n'))
              i += 1
          word = One_Word()
          word.en = en
          word.chs = chs
          word.num = num
          words.append(word)
    return words
```

字符串前加 u，说明文本是 UTF-8 编码

定义词典词条类

strip()为删除开头及结尾指定字符

split("=")为按字符"="分割

```
def cha_xun(danci,words):
    flag = False
    chs = ""
    for word in words:
        if flag == True:
            break
        if danci == str(word.en):
            num = word.num
            chss = word.chs
            flag = True
            for chsa in chss:
                chs += chsa
                chs +="\n"
    return chs

def get_word():
    w_en = ent_cha.get()
    #print(w_en)
    chs = cha_xun(w_en,words)
    if chs == "":
        strs = "Not find word ->"+ w_en
        text_chs.set(strs)
    else:
        text_chs.set(chs)

words = read_file()

# 定义窗体设置
win = Tk()
win.title('英汉小词典')
win.geometry('600×480')
# 定义提示标签
lab_cha = Label(win,text = "查 询:",font = 'Times-20',\
                width = 8,height = 2)
# 定义输入文本框
ent_cha = Entry(win,font = 'Times-20',width = 40)
# 定义提示标签
lab_shiyi = Label(win,text = "解 释:",font = 'Times-20',\
                  width = 8,height = 2)
text_chs = StringVar()
# 定义解释文本框
ent_shiyi = Label(win,textvariable = text_chs,\
                  font = '宋体-20',bg = 'white',\
                  width = 40,height = 17)
```

```
lab_cha.grid(row = 0,column = 0)
ent_cha.grid(row = 0,column = 1,columnspan = 2)
lab_shiyi.grid(row = 1,column = 0)
ent_shiyi.grid(row = 1,column = 1,columnspan = 2)
# 定义空行（用标签Label占位）
labss = Label(win,text = "",font = 'Times-20',\
             width = 1,height = 1)
labss.grid(row = 2,column = 0)
# 定义查询按钮
btn_cha = Button(win,text = "翻译",command = get_word,\
              font = 'Times-20',width = 10,height = 1)
btn_cha.grid(row = 3,column = 1)
# 定义退出按钮
btn_cha = Button(win,text = "退出",command = exit,\
              font = 'Times-20',width = 10,height = 1)
btn_cha.grid(row = 3,column = 2)

win.mainloop()
```

程序运行结果如图 6.11 所示。

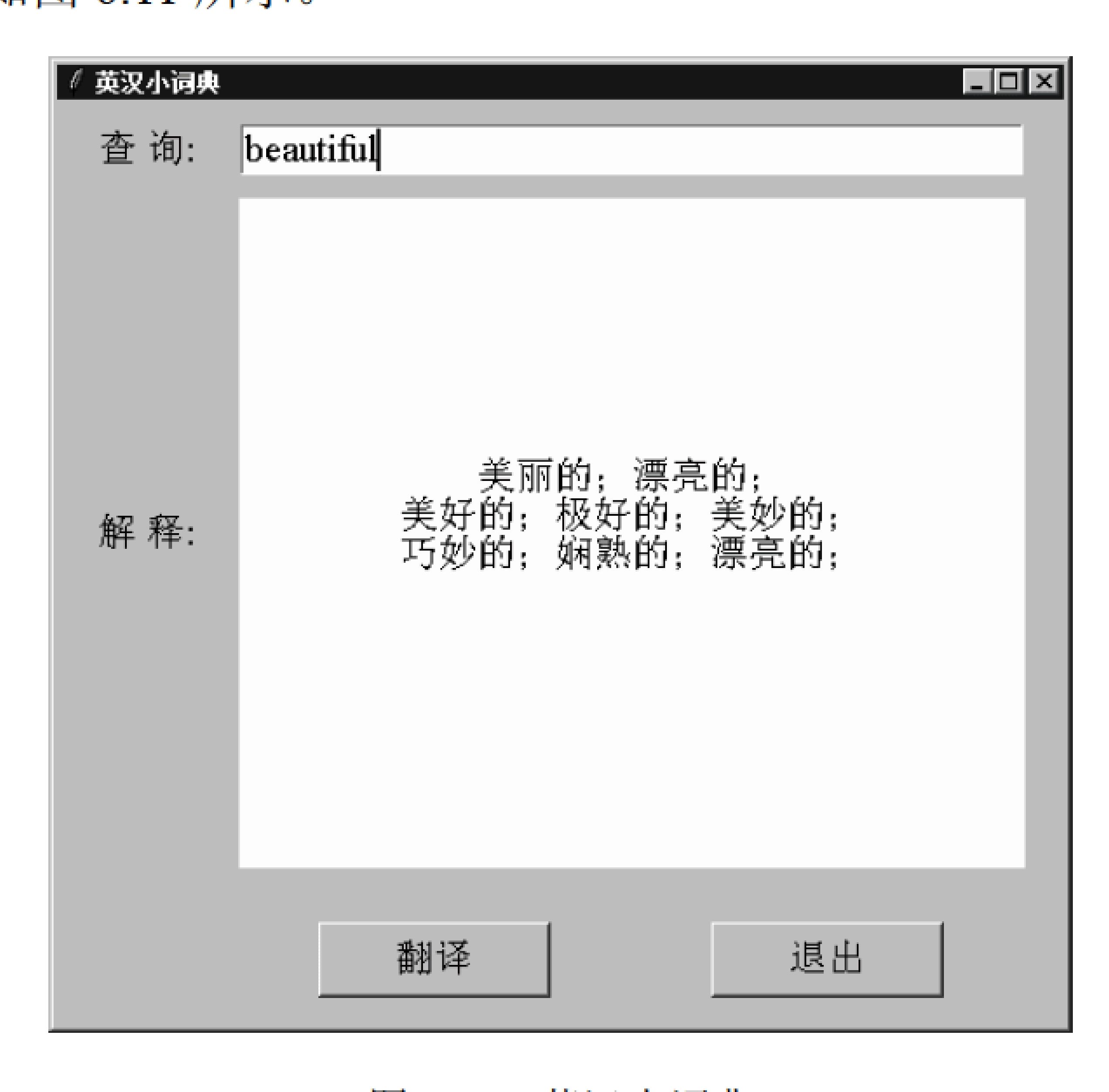


图 6.11　英汉小词典

习　题　6

1．编写程序，把 5 名同学的学号、姓名和成绩保存为二进制文件。

2．编写一个简易日记本，具有编辑和保存文件的功能。

3．编写一个输入学生成绩的窗体程序，具有保存数据为文件的功能，也能将数据导出到 Excel 文件中。

4．编写一个调用数据库的简易英汉字典程序，具有查询、添加、修改和删除单词的功能。

5．编写一个如图 6.12 所示的“简易商品管理系统”程序，对数据库中的商品具有查询、添加、修改等功能。



图 6.12　简易商品管理系统

第 7 章　多线程与异常处理



7.1　多线程编程

在应用程序中使用多线程编程，可以提高应用程序的处理速度和并发处理能力，使后台计算不影响前台界面与用户的交互操作。

7.1.1　线程与多线程

1. 线程

线程是指进程中单一顺序的执行流。设某程序的地址空间在 0x0000～0xffff，线程 A 运行在 0x2000～0x4000，线程 B 运行在 0x4001～0x6000，线程 C 运行在 0x6001～0x8000，0x8001～0xffff 为线程的公共数据区，多个线程共同构成一个大的进程，如图 7.1 所示。

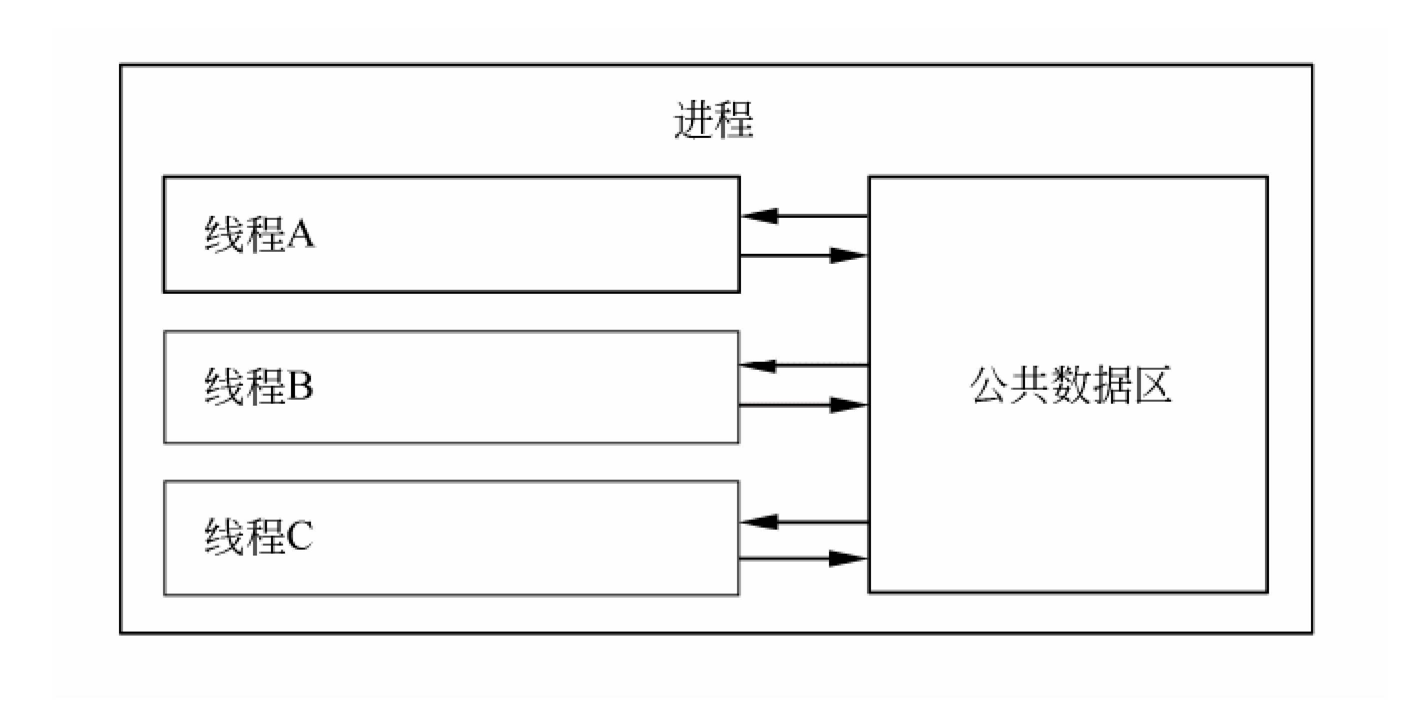


图 7.1　每个线程彼此独立，但有公共数据区

线程间的通信简单而有效，上下文切换非常快，它们是在同一个进程中的两部分之间进行的切换。每个线程彼此独立执行，一个程序可以同时使用多个线程来完成不同的任务。一般用户在使用多线程时并不需要考虑底层处理的细节。

2. 多线程

多线程程序是指一个程序中包含多个执行流，多线程是实现并发机制的一种有效手段。

例如，在传统的单进程环境下，用户必须等待一个任务完成后才能进行下一个任务，即使大部分时间空闲，也只能按部就班的工作，而多线程可以避免引起用户的等待。

又如，传统的并发服务器是基于多线程机制的，每个客户需要一个进程，而进程的数目是受操作系统限制的。基于多线程的并发服务器，每个客户一个线程，多个线程可以并

发执行。

进程与多线程的区别，如图 7.2 所示。

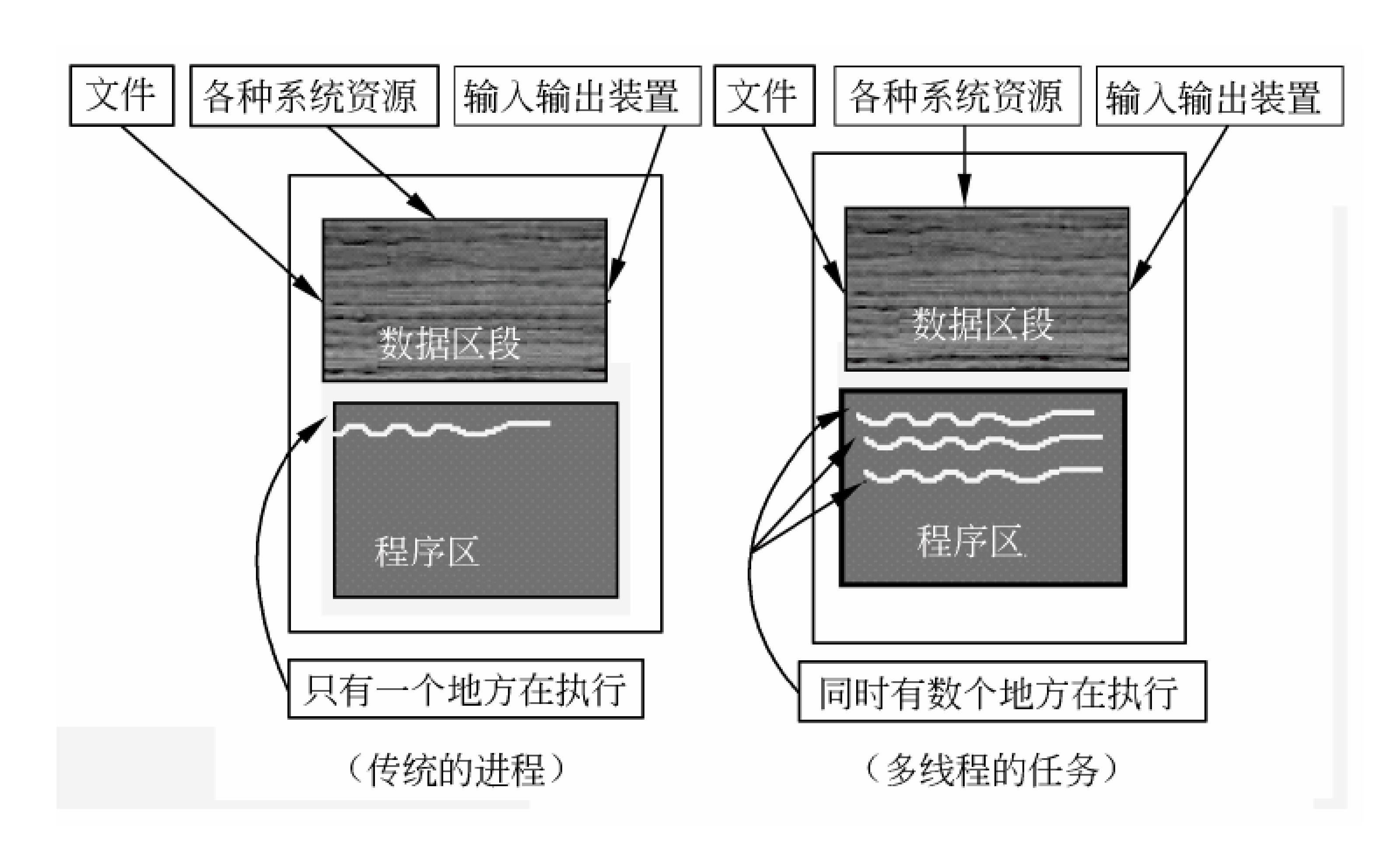


图 7.2　进程与线程的区别

从图 7.2 中可以看到，多任务状态下各进程的内部数据和状态都是完全独立的，而多线程共享一块内存空间和一组系统资源。

7.1.2　线程的生命周期

每个线程都要经历创建、就绪、运行、阻塞和死亡 5 个状态，线程从产生到消失的状态变化过程称为生命周期，如图 7.3 所示。

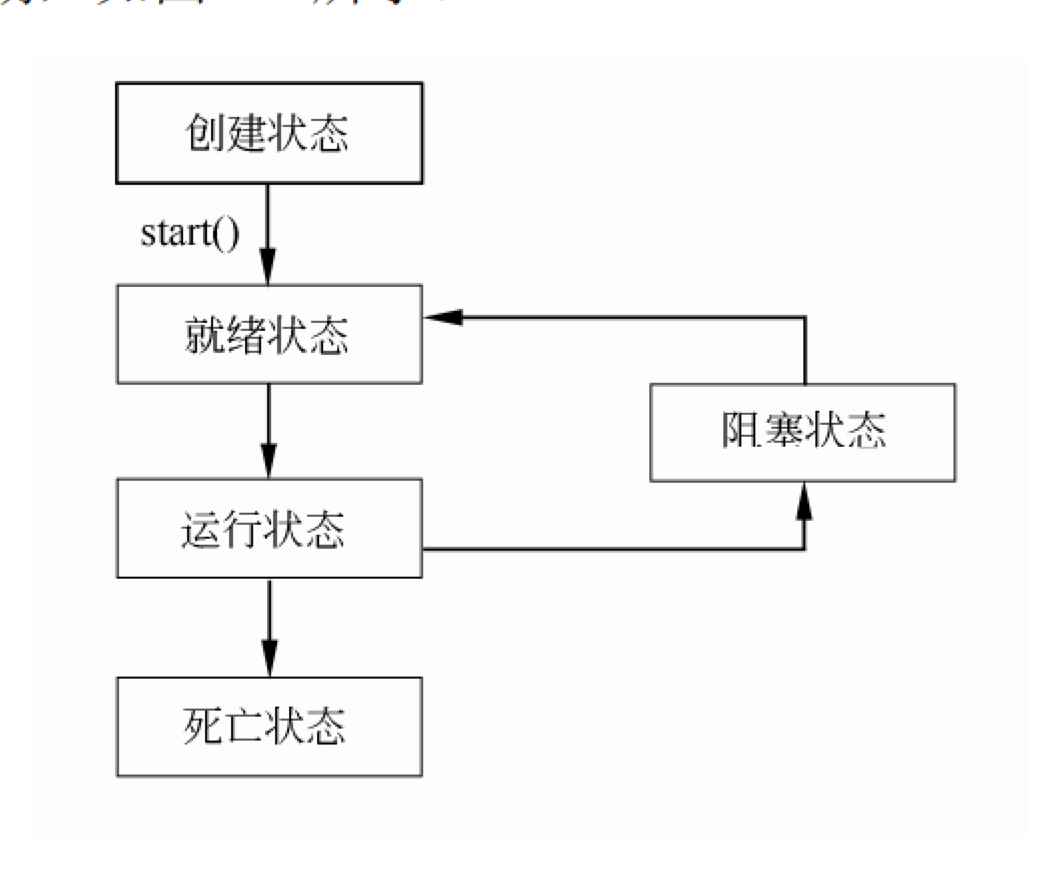


图 7.3　线程的生命周期

从图 7.3 可以看到，一个线程的生命周期一般经过如下步骤：

① 一个线程通过实例化创建线程对象后，进入新生状态。

② 线程对象通过调用 start()方法进入就绪状态，一个处在就绪状态的线程将被调度执行，执行该线程相应的 run()方法中的代码。

③ 通过调用线程（或从 Object 类继承）的 sleep()或 wait()方法，这个线程进入阻塞状态。一个线程也可能自己完成阻塞操作。

④ 当 run()方法执行完毕，或有一个例外产生，或执行 System.exit()方法，则一个线程就进入死亡状态。

7.1.3 创建线程的 threading.Thread 类

Python 通过引用 threading 模块创建多线程。在 threading 模块中定义了一个 Thread 类，进而创建线程。

threading.Thread 类的常用方法如表 7.1 所示。

表 7.1 threading.Thread 类的常用方法

方法	说明
__init__(self, name = threadname)	初始化方法，threadname 为线程的名称
run()	编写线程的代码，实现线程要完成的功能
getName()	获得线程对象名称
setName()	设置线程对象名称
start()	启动线程
jion([timeout])	等待另一线程结束后再运行
setDaemon(bool)	设置子线程是否随主线程一起结束，必须在 start()之前调用。默认为 False
isAlive()	检查线程是否在运行中

在 Python 中，可采用两种方式来创建线程：

① 通过创建 Thread 类的子类来构造线程，并重写它的 run()方法。

② 应用 Thread 类的构造函数创建一个多线程对象。

下面分别举例说明。

1. 应用 Thread 类的构造函数创建多线程

Thread 类的构造函数为：

```
class threading.Thread(group=None,target=None,name=None,args=(),kwargs={})
```

其中：

- group 恒为 None，保留未来使用；
- target 为要执行的函数，默认应为 None，意味着没有对象被调用；
- name 为线程名字。默认为 Thread-N；
- args 为要传入的参数，默认为()；
- Kwargs 为传入的参数为字典，默认为{}。

【例 7-1】 应用 Thread 类的构造函数创建多线程示例。

程序代码如下：

```
import threading        导入 threading 模块

def fun(i):
    print("thread id = %d \n" %i)

def main():
```

```
    for i in range(1,10):
        t = threading.Thread(target=fun, args=(i,))
        t.start()

if __name__ == "__main__":
    main()
```

创建线程对象

启动线程

程序运行结果如下：

```
thread id = 1
thread id = 2
thread id = 5
thread id = 3
thread id = 6
thread id = 4
thread id = 9
thread id = 8
thread id = 7
```

2. 创建 Thread 子类构造线程

threading.Thread 的子类必须重写父类的__init__()和 run()方法，并且在子类的__init__()方法中，要调用父类的__init__()方法。

【例 7-2】 创建 Thread 子类构造线程的示例。

程序代码如下：

```
import threading
import time

# 定义线程子类
class MyThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

    def run(self):
        print("starting", self.name)

def main():
    t1 = MyThread()
    t1.start()
    t2 = MyThread()
    t2.start()

if __name__ == "__main__":
    main()
```

线程子类包含__init__()和 run()方法

创建并启动线程对象 t1

创建并启动线程对象 t2

程序运行结果如下：

```
starting Thread-1
starting Thread-2
```

3. 比较两种创建线程对象

下面通过一个示例来说明上述两种线程对象的区别。

【例 7-3】 下面用 Thread 子类程序来模拟航班售票系统，实现两个售票窗口发售某班次航班的 10 张机票，一个售票窗口用一个线程来表示。

程序代码如下：

```
import threading
import time

# 定义线程子类
class MyThread(threading.Thread):
    tickets = 10
    def __init__ (self):
      threading.Thread.__init__ (self)

    def run(self):
      while(1):
        if(self.tickets>0):
          self.tickets = self.tickets-1
          print(self.name,"售机票售出第",self.tickets, " 号")
        else:
          exit()

def main():
    t1 = MyThread()
    t1.start()
    t2 = MyThread()
    t2.start()

if __name__ == "__main__":
    main()
```

线程需要完成的任务都放在run()方法中

程序运行结果如下：

```
Thread-1 售机票售出第 9 号
Thread-1 售机票售出第 8 号
Thread-1 售机票售出第 7 号
Thread-2 售机票售出第 9 号
Thread-1 售机票售出第 6 号
Thread-2 售机票售出第 8 号
Thread-1 售机票售出第 5 号
Thread-2 售机票售出第 7 号
Thread-1 售机票售出第 4 号
Thread-2 售机票售出第 6 号
Thread-1 售机票售出第 3 号
Thread-2 售机票售出第 5 号
Thread-1 售机票售出第 2 号
```

```
Thread-2 售机票售出第 4 号
Thread-1 售机票售出第 1 号
Thread-1 售机票售出第 0 号
Thread-2 售机票售出第 3 号
Thread-2 售机票售出第 2 号
Thread-2 售机票售出第 1 号
Thread-2 售机票售出第 0 号
```

【程序说明】

从运行结果中可以看到，每张机票被卖了两次，即两个线程各自卖 10 张机票，而不是去卖共同的 10 张机票。为什么会这样呢？这里需要的是，多个线程去处理同一资源，一个资源只能对应一个对象。在上面的程序中，创建了两个 Thread 对象，每个 Thread 对象中都有 10 张机票，每个线程都在独立地处理各自的资源。

【例 7-4】 下面用 Thread 类的构造函数创建的线程程序来模拟航班售票系统，实现两个售票窗口发售某班次航班的 10 张机票，一个售票窗口用一个线程来表示。

程序代码如下：

```
import threading

global tickets
tickets= 11
def fun(i):
    global tickets
    while(tickets>1):
        tickets = tickets-1
        print("第", i,"售机票窗口售出第",tickets, " 号")

def main():
    for i in range(1,3):
        t = threading.Thread(target=fun, args=(i,))
        t.start()

if __name__ == "__main__":
    main()
```

程序运行结果如下：

```
第 1 售机票窗口售出第 10 号
第 1 售机票窗口售出第 9  号
第 1 售机票窗口售出第 8  号
第 1 售机票窗口售出第 6  号
第 2 售机票窗口售出第 7  号
第 1 售机票窗口售出第 5  号
第 2 售机票窗口售出第 4  号
第 1 售机票窗口售出第 3  号
第 2 售机票窗口售出第 2  号
第 1 售机票窗口售出第 1  号
```

【程序说明】

在上面的程序中，创建了两个线程，每个线程调用的是同一个 Thread 对象中的 fun()方法，访问的是同一个对象中的变量（tickets）的实例。因此，这个程序能满足实际的要求。

7.1.4 线程同步

1. 多线程使用不当造成的数据混乱

多线程使用不当可能造成数据混乱。例如，两个线程都要访问同一个共享变量，一个线程读这个变量的值并在这个值的基础上完成某些操作，但就在此时，另一个线程改变了这个变量值，但第一个线程并不知道，这可能造成数据混乱。下面模拟两个用户从银行取款的操作造成数据混乱的一个例子。

【例 7-5】 设计一个模拟用户从银行取款的应用程序。设某银行账户存款额的初值是 2000 元，用线程模拟两个用户从银行取款的情况。

程序代码如下：

```
import threading
import time

# 定义银行账户类
class Mbank:
    global sum
    sum=2000
    def take(k):
        global sum
        temp=sum
        temp=temp - k
        time.sleep(0.2)
        sum = temp
        print("sum=",sum)
```

定义银行账户

```
# 模拟用户取款的线程子类
class MyThread(threading.Thread):
    tickets = 10
    def __init__ (self):
        threading.Thread.__init__ (self)

    def run(self):
        for i in range(1,5):
            Mbank.take(100)

def main():
    t1 = MyThread()
```

模拟用户取款，在线程中调用银行账户

```
    t1.start()
    t2 = MyThread()
    t2.start()

if __name__ == "__main__":
    main()
```

程序运行结果如下：

```
sum= 1900
sum= 1900
sum= 1800
sum= 1800
sum= 1700
sum= 1700
sum= 1600
sum= 1600
```

【程序说明】

该程序的本意是通过两个线程分多次从一个共享变量中减去一定数值，以模拟两个用户从银行取款的操作。

（1）类 Mbank 用来模拟银行账户，其中全局变量 sum 为账户现有存款额，take()方法表示取款操作， take()方法中的参数 k 表示每次的取款数。为了模拟银行取款过程中的网路阻塞，让系统休眠一随机时间段，再来显示最新存款额。

（2）MyThread 是模拟用户取款的线程类，在 run()方法中，通过循环 4 次调用 Mbank 类的方法 take()，从而实现分 4 次从存款额中取出 400 元的功能。

（3）main()方法启动创建两个 MyThread 类的线程对象，模拟两个用户从同一账户中取款。

账户现有存款额 sum 的初值是 2000 元，如果每个用户各取出 400 元，存款额的最后余额应该是 1200 元。但程序的运行结果却并非如此，并且运行结果是随机的，每次互不相同。

之所以会出现这种结果，是由于线程 t1 和 t2 的并发运行引起的。例如，当 t1 从存款额 sum 中取出 100 元，t1 中的临时变量 temp 的初始值是 2000 元，取款后将 temp 的值改变为 1900，在将 temp 的新值写回 sum 之前，t1 睡眠了一段时间。正在 t1 睡眠的这段时间内，t2 来读取 sum 的值，其值仍然是 2000，然后将 temp 的值改变为 1900，在将 temp 的新值写回 sum 之前，t2 睡眠了一段时间。这时，t1 睡眠结束，将 sum 更改为其 temp 的值 1900，并输出 1900。接着进行下一轮循环，将 sum 的值改为 1800，并输出后，再继续循环，在将 temp 的值改变为 1700 之后，还未来得及将 temp 的新值写回 sum 之前，t1 进入睡眠状态。这时，t2 睡眠结束，将它的 temp 的值 1900 写入 sum 中，并输出 sum 的现在值 1900……，如此继续，直到每个线程结束，出现了和原来设想不符合的结果。

通过对该程序的分析，发现出现错误结果的根本原因是两个并发线程共享同一内存变量所引起的。后一线程对变量的更改结果覆盖了前一线程对变量的更改结果，造成数据混乱。为了防止这种错误的发生，Python 提供了一种简单而又强大的同步机制。

2. 线程同步的方法

使用同步线程是为了保证在一个进程中多个线程能协同工作，所以线程的同步很重

要。所谓线程同步就是在执行多线程任务时，一次只能有一个线程访问共享资源，其他线程只能等待，只有当该线程完成自己的执行后，另外的线程才可以进入。

使用 Thread 对象的 Lock 和 Rlock 可以实现简单的线程同步，这两个对象都有 acquire()方法和 release()方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到 acquire()和 release()方法之间。

【例 7-6】 改写例 7-5，用线程同步的方法设计用户从银行取款的应用程序。

程序代码如下：

```
import threading
import time

threadLock = threading.Lock()  # 创建一个锁对象

# 定义银行账户类
class Mbank:
    global sum
    sum=2000
    def take(k):
        global sum
        temp=sum
        temp=temp - k
        time.sleep(0.2)
        sum = temp
        print("sum=",sum)


# 模拟用户取款的线程子类
class MyThread(threading.Thread):
    tickets = 10
    def __init__ (self):
        threading.Thread.__init__ (self)

    def run(self):
      for i in range(1,5):
        threadLock.acquire()  # 获得锁
        Mbank.take(100)
        threadLock.release()  # 释放锁

def main():
    t1 = MyThread()
    t1.start()
    t2 = MyThread()
    t2.start()
```

用户取款时使用线程同步，调用银行账户

```
if __name__ == "__main__":
    main()
```

程序运行结果如下：

```
sum= 1900
sum= 1800
sum= 1700
sum= 1600
sum= 1500
sum= 1400
sum= 1300
sum= 1200
```

【程序说明】

例 7-6 与例 7-5 比较，只是在线程类的 run()方法中增加了同步锁。由于对 take()方法增加了同步限制，所以在线程 t1 结束 take()方法运行之前，线程 t2 无法进入 take()方法。同理，在线程 t2 结束 take()方法运行之前，线程 t1 无法进入 take()方法。从而避免了一个线程对 sum 变量的更改结果覆盖了另一线程对 sum 变量的更改结果。

视频录像

7.2 异常处理

异常（Exception）指程序运行过程中出现的非正常现象，如用户输入错误、需要处理的文件不存在、在网络上传输数据但网络没有连接等。由于异常情况总是可能发生的，良好健壮的应用程序除了具备用户要求的基本功能外，还应该具备预见并处理可能发生各种异常的功能。所以，开发应用程序时要充分考虑到各种可能发生的异常情况，使程序具有较强的容错能力。把这种对异常情况进行技术处理的技术称为异常处理。

7.2.1 Python 中的常见标准异常

Python 系统定义了一系列可能发生的异常类型，这些预定义的异常类型称为标准异常。当程序运行过程中发生标准异常时系统会显示相应的提示信息。在 Python 系统中定义的常见标准异常如表 7.2 所示。

表 7.2　Python 中定义的常见标准异常

异常类型	说明
BaseException	所有异常的基类
KeyboardInterrupt	用户中断执行（通常是输入^C）
Exception	常规错误的基类
StandardError	所有的内建标准异常的基类
FloatingPointError	浮点计算错误
OverflowError	数值运算超出最大限制
ZeroDivisionError	除（或取模）零（所有数据类型）

续表

异常类型	说明
AttributeError	对象没有这个属性
IOError	输入输出操作失败
WindowsError	系统调用失败
ImportError	导入模块/对象失败
UnboundLocalError	访问未初始化的本地变量
SyntaxError	Python 语法错误
IndentationError	缩进错误

在 Python 系统中，还有许多类型的异常，在这里就不一一列举了。

7.2.2 异常的捕捉与处理

在 Python 中，使用 try 语句实现捕捉与处理异常。try 语句有多种形式，现介绍如下：

1. 使用 try…except 语句

try…except 语句的语法格式为：

```
try:
    <被检测的语句块>
except <异常类型名称>:
    <处理异常的语句块>
```

语句检测 try 语句后面的<被检测的语句块>中是否有异常，如果有异常，则执行 except 语句后面的<处理异常的语句块>中的语句；否则，直接忽略<处理异常的语句块>，执行后续语句。

【例 7-7】 元组下标越界引发异常。

程序代码如下：

```
s=[1,2,3,4,5]
try:
    print(s[5])
except IndexError:
    print("发生异常原因：索引出界")
```

运行程序结果如下：

```
发生异常原因：索引出界
```

2. 使用 try…except…else 语句

try…except…else 语句的语法格式为：

```
    try:
    <被检测的语句块>
except <异常类型名称>:
    <处理异常的语句块>
```

```
 Else:
    <无异常时执行的语句块>
```

语句检测 try 语句后面的<被检测的语句块>中是否有异常，如果有异常，则执行 except 语句后面的<处理异常的语句块>中的语句；否则，忽略<处理异常的语句块>，直接执行 else 语句。

【例 7-8】 编写一个把字符串写入一个文件的程序。若写入成功，则提示“内容写入文件成功”，否则提示“Error：没有找到文件或读取文件失败”。

程序代码如下：

```
try:
    fh = open("testfile.txt", "w")
    fh.write("这是一个测试文件,用于测试异常!!")
except IOError:
    print("Error: 没有找到文件或读取文件失败")
else:
    print("内容写入文件成功")
    fh.close()
```

3. 带有多个 except 子句的 try 语句

带有多个 except 子句的 try 语句的语法格式与 try…except 语句的语法格式相似，只是使用了多个 except 子句。

【例 7-9】 编写程序，从键盘输入 1，2，…，5 中的一个数字；当输入其他数字或字符则提示为异常。

程序代码如下：

```
s=[1,2,3,4,5]
while True:
    try:
        i = eval(input('input:'))
        print(s[i])
    except IndexError:
        print("发生异常原因：索引出界")
        break
    except NameError:
        print("发生异常原因：不是数字")
        break
    except KeyboardInterrupt:
        print("发生异常原因：用户中断输入")
        break
    else:
        pass
```

【程序说明】这段代码的功能是：循环读取键盘输入的数据，作为索引输出列表的值；被检测的代码块可能有三种异常，它们是索引越界、输入的是字符而不是数字、用户中断了数据输入；无论哪种异常发生都会终止循环并结束程序。

4. 带有 finally 子句的 try 语句

在 try 语句中，如果带有 finally 子句，则无论异常是否发生，finally 子句均会被执行。

【例 7-10】 带有 finally 子句的 try 语句示例。

程序代码如下：

```
s=input("请输入你的年龄:")
if s =="":
    raise Exception("输入不能为空。")
try:
    i=int(s)
except Exception as err:
    print(err)
finally:
    print("Goodbye!")
```

raise 为引发异常的语句

7.3 正则表达式



视频录像

7.3.1 字符匹配与匹配模式

1. 字符匹配

假设要搜索一个包含字符 cat 的字符串，搜索用的子字符串就是 cat。如果搜索对大小写不敏感，单词 catalog、Catherine、sophisticated 都可以匹配。也就是说：

子字符串：cat。

匹配：catalog、Catherine、sophisticated。

2. 匹配模式

例如，使用“?”和“*”通配符来查找硬盘上的文件。“?”通配符匹配文件名中的一个字符，而“*”通配符匹配多个字符。这时，“?”和“*”通配符就是一种匹配模式。

例如，“data?.dat”这样的模式将查找下列文件：

```
data.dat
data1.dat
data2.dat
datax.dat
dataN.dat
```

若使用“*”字符代替“?”字符扩大了找到的文件的数量。data*.dat 匹配下列所有文件：

```
data.dat
data1.dat
data2.dat
data12.dat
datax.dat
dataXYZ.dat
```

7.3.2 正则表达式的规则

1. 构建正则表达式规则的特殊字符

正则表达式是一种可以用于模式匹配和替换的一种逻辑公式，就是用事先定义的一些特定字符及这些特定字符的组合，组成一个“规则字符串”，这个“规则字符串”用来表达对字符串的一种过滤逻辑。一个正则表达式就是由普通的字符（如字符'a'～'z'）以及特殊字符（称为“元字符”）组成的文字模式。该模式用以描述在查找文字主体时待匹配的一个或多个字符串。

正则表达式的使用，可以通过简单的办法来实现强大的功能。

下面先看一个用特殊字符（元字符）表示正则表达式规则的示例：

```
^ [ 0 - 9 ] + abc$
```

其中：

- ^：匹配字符串的开始位置。
- [0-9]+：匹配多个数字，‘[0-9]’匹配单个数字，‘+’匹配一个或多个。
- abc$：匹配字母 abc 并以“abc”结尾，‘$’为匹配输入字符串的结束位置。

该规则表示，需要匹配以数字开头并以“abc”结尾的字符串。

【例 7-11】 编写程序，匹配以数字开头并以“abc”结尾的字符串。

程序代码如下：

```
import re

str = r"123abc"                    # 需要匹配的源文本
p1 = r"^[0-9]+abc$"                # 编写正则表达式规则
patt1 = re.compile(p1)             # 编译正则表达式
matc1 = re.search(patt1, str)      # 在源文本中搜索符合正则表达式的部分
print(matc1.group(0))              # 获得分组信息并输出匹配结果
```

程序运行结果如下：

```
123abc
```

构建正则表达式规则的常用特殊字符（元字符）如表 7.3 所示。

表 7.3 构建正则表达式规则的常用特殊字符

元字符	说明
\	反斜杠，转义符
^	限制开头字符，如“^python”表示限制以 python 为开头字符
$	限制结尾字符，如“python$”表示限制以 python 为结尾字符

续表

元字符	说明
[ ]	只有方括号中指定的字符才参与匹配，如[ab]表示匹配字符 a 或 b
[^]	方括号中的 ^ 为否符号，如[^a]表示匹配以字母 a 开头除外的字符
.	通配符，代表任意一个单独字符。使用时，需要用“\.”表示
{n,}	n 是一个非负整数，至少匹配 n 次。例如，“o{2,}”不能匹配 Bob 中的 o，但能匹配 foooood 中的所有 o
*	匹配前面的子表达式任意次。例如，zo*能匹配 z，也能匹配 zo 以及 zoo，‘*’ 等价于 o{0,}
+	匹配前面的子表达式至少一次。例如，zo+能匹配 zo 以及 zoo，但不能匹配 z，“+” 等价于{1,}

2. 使用正则表达式进行匹配的流程

正则表达式的匹配过程是：依次比较表达式和文本中的字符，如果每一个字符都能匹配，则匹配成功；一旦有匹配不成功的字符则匹配失败。

使用正则表达式进行匹配的流程如图 7.4 所示。

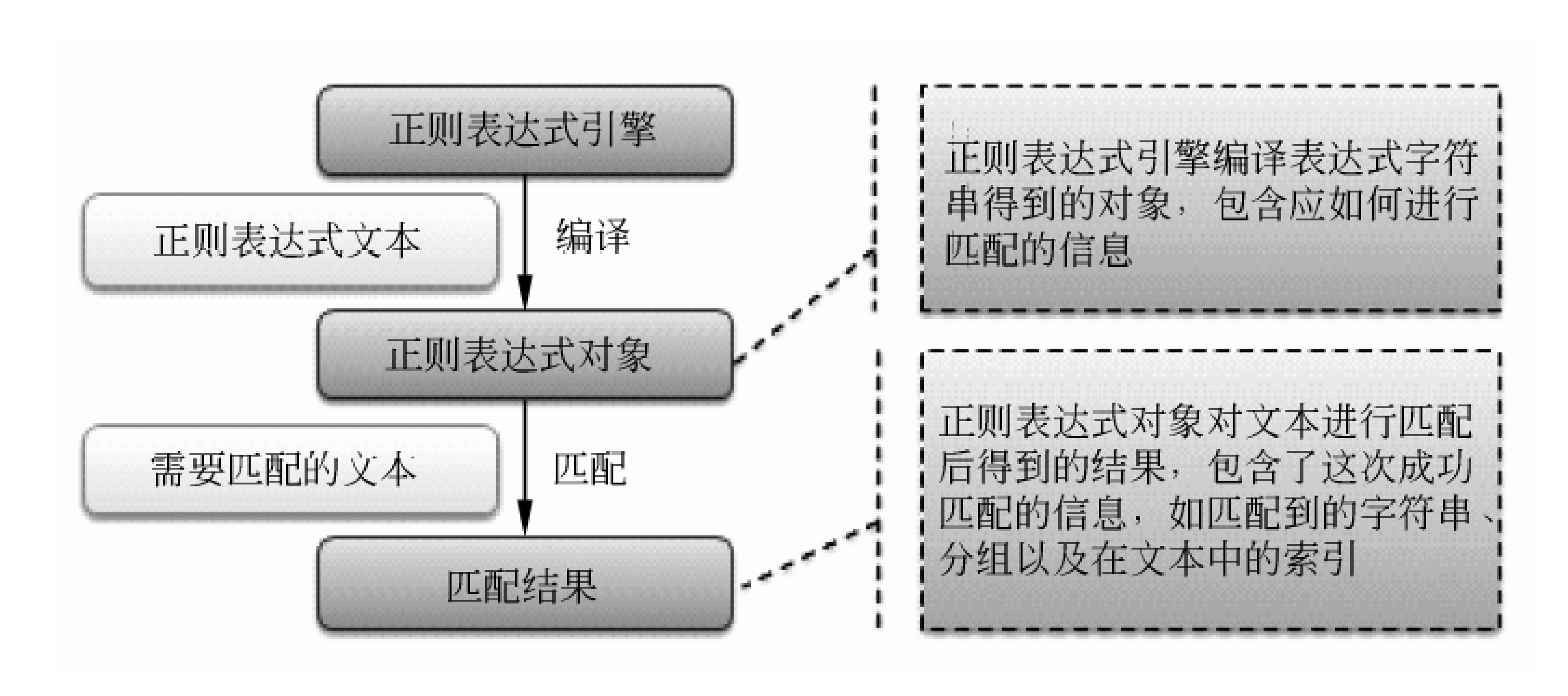


图 7.4　使用正则表达式进行匹配的流程

7.3.3　正则表达式 re 模块的方法

1. re 模块的方法

正则表达式 re 模块提供了正则表达式操作需要的方法，利用这些方法，可以很方便地得到匹配结果。

正则表达式 re 模块的常用方法如表 7.4 所示。

表 7.4　re 模块的常用方法

方法	说明
compile(pattern)	编译正则表达式，创建模式对象
search(pattern,string)	在字符串中寻找模式，返回 match 对象
match(pattern,string)	在字符串开始处匹配模式
split(pattern,string)	根据模式分隔字符串，返回列表
findall(pattern,string)	以列表形式返回匹配项

2. 模式对象的方法

构建的模式对象有几个常用方法，现介绍如下。

（1）group()

group()用于获取子模式（组）的匹配项。

例如：

```
pat = re.compile(r'www\.(.*)\.(.*)')   # 用()表示一个组，这里两个组
m = pat.match('www.dxy.com')
m.group()         # 默认值为0，表示匹配整个字符串，返回'www.dxy.com'
m.group(1)        # 返回给定组1匹配的子字符串'dxy'
m.group(2)        # 返回给定组2匹配的子字符串'com'
```

（2）start()

start()为指定组匹配项的开始位置。

例如：

```
m.start(2)        # 组2开始的索引，返回值为8
```

（3）end()

end()为指定组匹配项的结束位置。

例如：

```
m.end(2)          # 组2结束的索引，返回值为11
```

【例 7-12】 编译正则表达式，创建模式对象示例。

程序代码如下：

```
import re
patt1 = re.compile('A')          # 编译正则表达式，patt1为模式对象
matc1 = patt1.search('CBA')      # 等价于 re.search('A','CBA')
print(matc1)
# 匹配，返回<_sre.SRE_Match object : span=(2, 3), match='A'>

matc2 = pat t1.search('CBD')
print(matc2)
# 没有匹配，返回None(False)
```

【例 7-13】 在一个字符串中查找子串示例。

程序代码如下：

```
# 第一步，要引入re模块
import re
# 第二步，调用模块函数
a = re.findall("创造", "创新的目的是为了给社会创造更大的价值")
print(a)     # 以列表形式返回匹配到的字符串
```

程序运行结果如下：

```
[创造']
```

【例 7-14】 编写程序，把文件 a.txt 里的字符串 Hello 替换成“你好”，并把更换后的文件保存到 b.txt。

程序代码如下：

```
import re

f1 = open('a.txt', 'r+')
f2 = open('b.txt', 'w+')

str1 = r'Hello'    #前缀r表示“自然字符串”,特殊字符失去意义
str2 = r'你好'

for s in f1.readlines():
    tt = re.sub(str1, str2, s)
    f2.write(tt)

f1.close()
f2.close()
```

运行程序后，在当前目录下生成一个 b.txt 文件，文件 a.txt 和替换后文件 b.txt 的内容如图 7.5 所示。



（a）替换前文件 a.txt 的内容



（b）替换后文件 b.txt 的内容

图 7.5　用“你好”替换 Hello

7.4　案 例 精 选



视频录像

【例 7-15】 应用多线程，编写一个“幸运大转盘”抽奖游戏程序。

程序代码如下：

```
import tkinter
import time
import threading

root = tkinter.Tk()
root.title('"幸运大转盘"抽奖游戏')
```

```
root.minsize(300,300)

btn1 = tkinter.Button(text = '奔驰',bg = 'red')
btn1.place(x = 20, y = 20, width = 50, height = 50)

btn2 = tkinter.Button(text = '宝马',bg = 'white')
btn2.place(x = 90, y = 20, width = 50, height = 50)

btn3 = tkinter.Button(text = '奥迪',bg = 'white')
btn3.place(x = 160, y = 20, width = 50, height = 50)

btn4 = tkinter.Button(text = '日产',bg = 'white')
btn4.place(x = 230, y = 20, width = 50, height = 50)

btn5 = tkinter.Button(text = '宾利',bg = 'white')
btn5.place(x = 230, y = 90, width = 50, height = 50)

btn6 = tkinter.Button(text = '劳斯',bg = 'white')
btn6.place(x = 230, y = 160, width = 50, height = 50)

btn7 = tkinter.Button(text = '奇瑞',bg = 'white')
btn7.place(x = 230, y = 230, width = 50, height = 50)

btn8 = tkinter.Button(text = '吉利',bg = 'white')
btn8.place(x = 160, y = 230, width = 50, height = 50)

btn9 = tkinter.Button(text = '大众',bg = 'white')
btn9.place(x = 90, y = 230, width = 50, height = 50)

btn10 = tkinter.Button(text = '沃尔沃',bg = 'white')
btn10.place(x = 20, y = 230, width = 50, height = 50)

btn11 = tkinter.Button(text = '红旗',bg = 'white')
btn11.place(x = 20, y = 160, width = 50, height = 50)

btn12 = tkinter.Button(text = '长城',bg = 'white')
btn12.place(x = 20, y = 90, width = 50, height = 50)

# 将所有选项组成列表
carlist = [btn1,btn2,btn3,btn4,btn5,btn6,btn6,btn7,btn8, btn9,btn10,btn11,
btn12]
# 是否开始循环的标志
isloop = False
def round():
    # 判断是否开始循环
```

```
    if isloop == True:
        return
    # 初始化计数变量
    i = 0
    # 死循环
    while True:
        time.sleep(0.1)
        # 将所有的组件背景变为白色
        for x in carlist:
            x['bg'] = 'white'
        # 将当前数值对应的组件变色
        carlist[i]['bg'] = 'red'
        i += 1    # 变量+1
        # 如果i大于最大索引直接归零
        if i >= len(carlist):
            i = 0
        if functions == True :
            continue
        else :
            break

# “开始”按钮事件：建立一个新线程的函数
def newtask():
    global isloop
    global functions
    # 建立新线程
    t = threading.Thread(target = round)
    # 开启线程运行
    t.start()
    # 设置程序开始标志
    isloop = True
    # 是否运行的标志
    functions = True

# "停止"按钮事件：停止循环
def stop():
    global isloop
    global functions

    functions = False
    isloop = False

#  开始/停止按钮
btn_start = tkinter.Button(root,text = '开始',command = newtask)
btn_start.place(x = 90, y = 120, width = 50, height = 50)
```

```
btn_stop = tkinter.Button(root,text = '停止',command = stop)
btn_stop.place(x = 160, y = 120, width = 50, height = 50)

root.mainloop()
```

程序运行结果如图 7.6 所示。

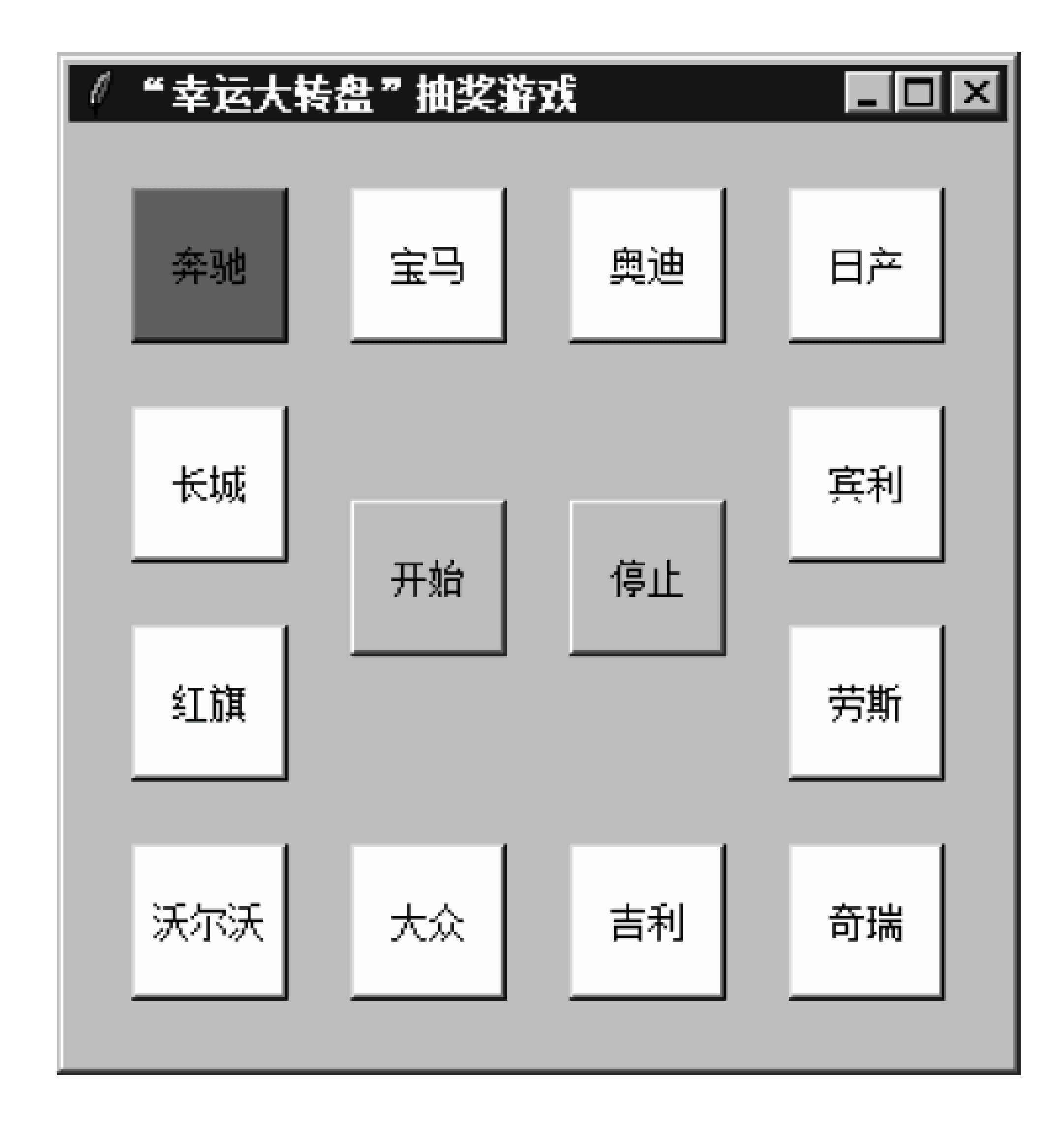


图 7.6 "幸运大转盘"抽奖游戏

习 题 7

1．编写一个龟兔赛跑的多线程程序，单击按钮后，龟兔开始赛跑（兔子比乌龟跑得速度快 5 倍，但休息的时间则长 10 倍）。

2．用多线程编写模拟自由落体与平抛运行的程序。

第 8 章　网络程序设计

网络应用的核心思想是使连入网络的不同计算机能够跨越空间协同工作，要求它们之间能够准确、迅速地传递信息。Python 是一门非常适合于分布计算环境的语言，网络应用是它的重要应用之一，尤其是它具有非常好的 Internet 网络程序设计功能。

本章将介绍 Python 用于编写网络通信及网络应用程序的设计方法。

8.1　套接字 Socket 编程基础



视频录像

8.1.1　套接字 Socket

1. 网络通信中的端口

由于一台计算机上可同时运行多个网络程序，IP 地址只能保证把数据信息送到该计算机，但无法知道要把这些数据交给该主机上的哪个网络程序，因此用“端口号”来标识正在计算机上运行的进程（程序）。每个被发送的网络数据包也都包含“端口号”，用于将该数据帧交给具有相同端口号的应用程序处理。

例如，在一个网络程序指定了自己所用的端口号为 52000，那么其他网络程序（如端口号为 13）发送给这个网络程序的数据包必须包含 52000 端口号，当数据到达计算机后，驱动程序根据数据包中的端口号，就知道要将这个数据包交给这个网络程序，如图 8.1 所示。

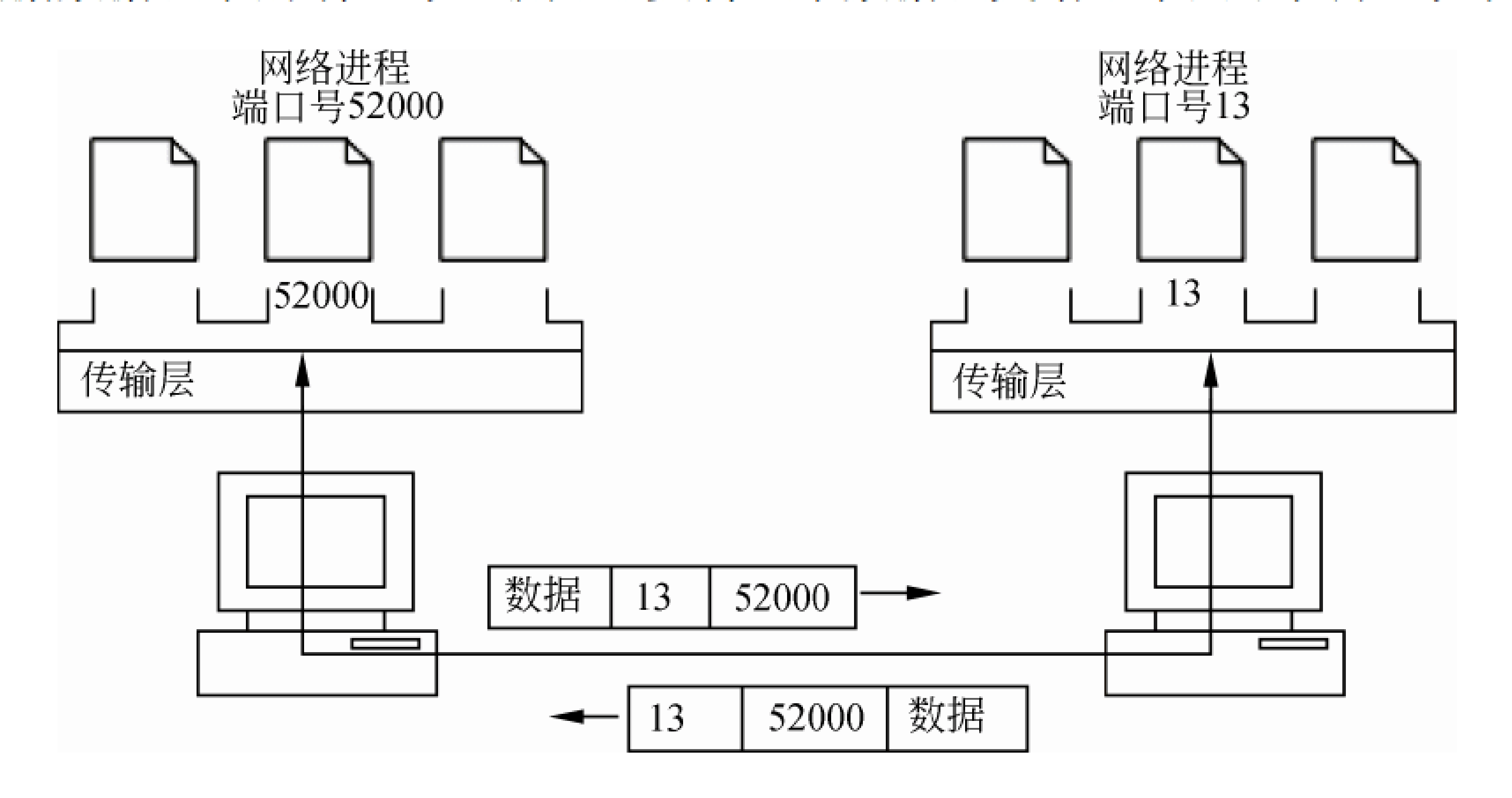


图 8.1　用“端口号”来标识进程

端口号是一个整数，其取值范围为 0～65535。由于同一台计算机上不能同时运行两个有相同端口号的进程。通常 0～1023 的端口号作为保留端口号，用于一些网络系统服务和应用，用户的普通网络应用程序应该使用 1024 以后的端口号，从而避免端口号冲突。

2. 套接字 Socket

在网络通信中，通过 IP 地址可以在网络上找到主机，通过端口可以找到主机上正在运行的网络程序。在 TCP/IP 通信协议中，套接字就是 IP 地址与端口号的组合。如图 8.2 所示，IP 地址 193.14.26.7 与端口号 13 组成一个套接字。

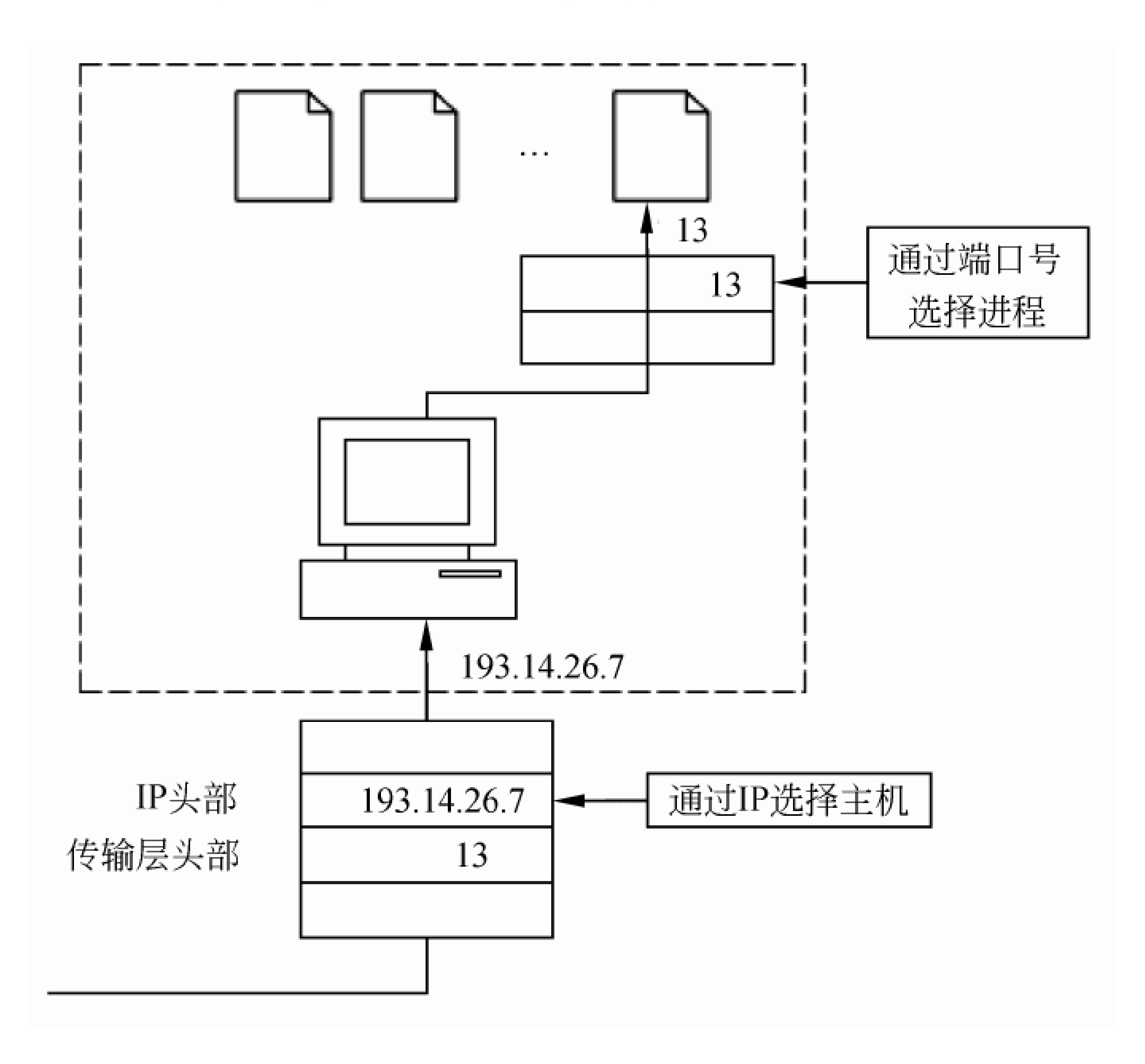


图 8.2　套接字是 IP 地址和端口号组合

Python 使用了 TCP/IP 套接字机制，并使用一些类来实现套接字中的概念。Python 中的套接字提供了在一台处理机上执行的应用程序与在另一台处理机上执行的应用程序之间进行连接的功能。

网络通信，准确地说，不仅是两台计算机之间在通信，而是两台计算机上执行的网络应用程序（进程）之间在收发数据。

当两个网络程序需要通信时，它们可以通过使用 Socket 类建立套接字连接。可以把套接字连接想象为一个电话呼叫，当呼叫完成后，通话的任何一方都可以随时讲话。但是在最初建立呼叫时，必须有一方主动呼叫，而另一方则正在监听铃声。这时，把呼叫方称为“客户端”，负责监听的一方称为“服务器端”。

8.1.2　TCP 与 UDP

在网络协议中，有两个高级协议是在网络应用程序编写中常用的，它们是传输控制协议（transmission control protocol，TCP）和用户数据报协议（user datagram protocol，UDP）。

TCP 是面向连接的通信协议，TCP 提供两台计算机之间可靠无差错的数据传输。应用程序利用 TCP 进行通信时，信息源与信息目标之间会建立一个虚连接。这个连接一旦建立成功，两台计算机之间就可以把数据当作一个双向字节流进行交换。接收方对于接收到的每一个数据包都会发送一个确认信息，发送方只有收到接收方的确认信息后才发送下一个数据包。通过这种确认机制保证数据传输无差错。

UDP 是无连接通信协议，UDP 不保证可靠数据的传输。简单地说，如果一个主机向另外一台主机发送数据，这一数据就会立即发送，而不管另外一台主机是否已准备接收数据。如果另外一台主机收到了数据，它不会确认收到与否。这一过程，类似于从邮局发送信件，无法确定收信人一定收到了发出去的信件。

8.2 套接字 Socket 程序设计

8.2.1 基于 TCP 的客户机/服务器模式



视频录像

Python 系统内部集成了 Socket 模块，编写网络套接字通信程序时，可以直接使用 Socket 模块。

利用 Socket 方式进行数据通信与传输，大致有如下步骤：

1. 创建服务器端套接字 Socket，监听客户端的连接请求

（1）通过 socket()函数创建服务器端套接字 Socket 对象。

（2）Socket 对象用 bind()函数把服务器的 IP 地址绑定到这个套接字上。

（3）Socket 对象用 listen()函数监听客户端的连接请求。

（4）Socket 对象用 accept()函数等待并接收客户端的连接，连接成功则创建一个新的通信套接字。

2. 创建客户端 Socket 对象，向服务器端发起连接

（1）通过 socket()函数创建客户端 Socket 对象。

（2）客户端 Socket 对象用 connect()函数发起连接请求。

（3）建立连接。

3. 客户机与服务器通信

（1）服务器通信套接对象用 sendall()函数向客户端发送信息。

（2）客户端套接字 Socket 对象用 recv()函数接收服务器发来的信息。

（3）客户端套接字 Socket 对象用 sendall()函数向服务器发送信息。

（4）服务器通信套接字对象用 recv()函数接收客户端发来的信息。

（5）通信完毕，使用 close()函数关闭套接字。

客户机/服务器模式的连接请求与响应过程如图 8.3 所示。

【例 8-1】 远程数据通信示例，本例由客户端程序和服务器程序两部分组成。

（1）服务器端程序 ex8_1_server.py

```
from socket import *

HOST = '127.0.0.1'
PORT = 4321
ADDR = (HOST, PORT)
ss = socket(AF_INET, SOCK_STREAM, 0)
ss.bind(ADDR)
ss.listen(10)
```

创建和设置套接字对象

```
while True:
    print('等待客户机连接……\n')
    cs, caddr = ss.accept()
    print('连接的客户机来自于：', caddr)
    str = '欢迎你访问本服务器！'
    cs.sendall(bytes(str, 'UTF-8'))
    msg = cs.recv(1024).decode()
    print('接收客户机信息：', msg)
    cs.close()

ss.close()
```

向客户机发送信息

接收客户机发来的信息

关闭套接字对象

（服务器端）
（客户机端）
创建Socket对象
创建客户机Socket
bind（地址，端口号）
监听客户机连接listen()
创建通信套接字accept()
127.0.0.1：4321
发起连接请求connect()
“欢迎你访问本服务器！”
向客户机发送信息
接收服务器端信息
接收客户机信息
向服务器端发送信息
“已收到服务器发来的信息！”
关闭套接字，断开连接
关闭套接字，断开连接

图 8.3　客户机/服务器模式

（2）客户端程序 ex8_1_client.py

```
from socket import *
import tkinter
from tkinter import scrolledtext     # 导入滚动文本框的模块

win = tkinter.Tk()
win.title('客户端程序')

# 创建一个容器
monty = tkinter.LabelFrame(win, text=" 接收信息 ")
                                     # 创建LabelFrame容器其父容器为win
```

```
monty.grid(column=0, row=0, padx=10, pady=10)
                                    # padx和pady为该容器外围需要留出的空余空间

# 滚动文本框
scrolW = 60                                # 设置文本框的长度
scrolH = 5                                 # 设置文本框的高度

txt_recv = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH)
txt_recv.grid(column=0, columnspan=3)      # columnspan将3列合并成一列

# 按钮收发信息事件
def conn():
  HOST = '127.0.0.1'
  PORT = 4321
  ADDR = (HOST, PORT)
  cs = socket(AF_INET, SOCK_STREAM, 0)
  cs.connect(ADDR)                         #向服务器发起连接请求

  data = cs.recv(1024).decode()      <—— 接收服务器发来的信息
  msgcontent = '接收到服务器发来的信息: \n '
  text_recv.insert('end', msgcontent + data, 'green') <— 在文本框显示接收的信息

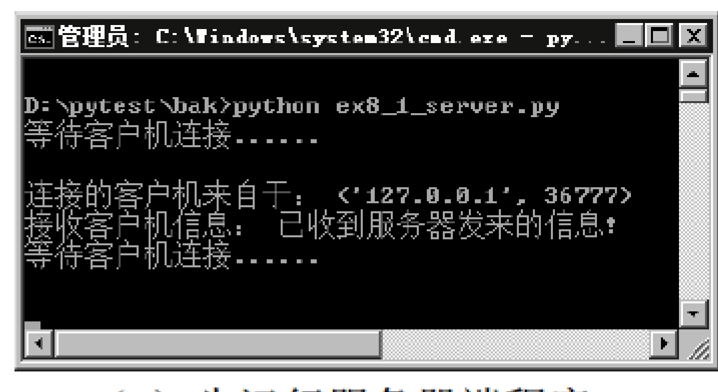
 str = '已收到服务器发来的信息!'           } 向服务器发送信息
 cs.sendall(bytes(str, 'UTF-8'))         }

 cs.close()

# 按钮对象
action = tkinter.Button(monty, text="连接服务器", command=conn)
action.grid(column=2, row=1)      # 设置布局位置，column代表列，row 代表行
# 主事件循环
win.mainloop()                    # 当调用mainloop()时,窗口才会显示出来
```

运行程序时，先打开一个控制台窗口，用命令执行服务器端程序后，再重新打开一个新的控制台窗口，用命令执行客户机程序。

程序运行结果如图 8.4 所示（先运行服务器端程序，再运行客户端程序）。



（a）先运行服务器端程序



（b）后运行客户机端程序

图 8.4　客户机/服务器通信

8.2.2　基于 UDP 的网络程序设计

UDP 是面向无连接的协议。使用 UDP 时，不需要建立连接，只要知道对方的 IP 地址和端口号，就可以直接发数据包。例如，发送短信，只要数据发送出去，无须了解对方是否接收到。

UDP 的编写步骤如下：

- 创建 Socket 套接字；
- 发送/接收数据；
- 关闭套接字。

【例 8-2】 编写程序，实现下列功能：

① 一台客户机从键盘输入一行字符，并通过其套接字将该行发送到服务器。

② 服务器从其连接套接字读取字符。

③ 服务器将该行字符转换成大写。

④ 服务器将修改后的字符串（行）通过连接套接字再发回给客户机。

⑤ 客户机从其套接字中读取修改后的行，然后将该行在监视器上显示。

（1）服务器端

程序代码如下：

```
import socket
#创建一个socket,SOCK_DGRAM表示UDP
s = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

s.bind(('127.0.0.1', 10021))        # 绑定IP地址及端口

print('Bound UDP on 10021...')

while True:
    # 获得数据和客户端的地址与端口,一次最大接收1024B
    data, addr = s.recvfrom(1024)
    print('Received from %s:%s.' % addr)
    # 将数据变成大写送回客户端
    s.sendto(data.decode('utf-8').upper().encode(), addr)

# 不关闭socket
```

（2）客户机端

程序代码如下：

```
#coding=utf-8

from socket import *

# 创建udp套接字
```

```
udp_socket = socket(AF_INET, SOCK_DGRAM)

# 准备接收方的地址
# 127.0.0.1表示目的IP地址
# 10021表示目的端口
dest_addr = ('127.0.0.1', 10021)  # 地址为元组，IP是字符串，端口是数字

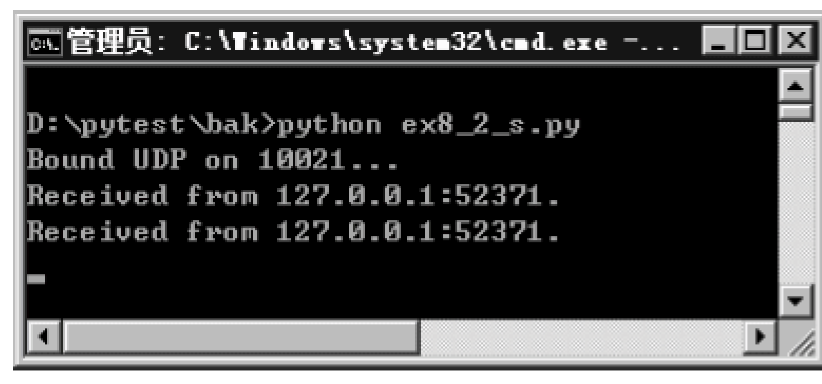
while True:
    # 从键盘获取数据
    send_data = input("请输入要发送的数据:")
    if not send_data or send_data == 'quit':
        break
    # 发送数据到指定计算机上的指定程序中
    udp_socket.sendto(send_data.encode('utf-8'), dest_addr)

    # 等待接收对方发送的数据
    # 如果没有收到数据则会阻塞等待，直到收到数据
    recv_data = udp_socket.recvfrom(1024)  # 1024表示本次接收的最大字节数

    # 显示对方发送的数据
    # 接收到的数据recv_data是一个元组
    # 第1个元素是对方发送的数据
    # 第2个元素是对方的ip和端口
    print(recv_data[0].decode('utf-8'), recv_data[1])

# 关闭套接字
udp_socket.close()
```

运行程序时，应该先运行服务器端程序，然后再运行客户机端程序，运行结果如图 8.5 所示。



（a）先运行服务器端程序



（b）后运行客户机端程序

图 8.5 UDP 通信示例

8.3 网络应用案例精选



视频录像

8.3.1 文件传输协议 FTP 应用

文件传输协议（file transfer protocol，FTP）是网络应用中最常用的一种协议。使用 FTP

传输文件时，需要使用 FTP 客户端程序登录到 FTP 服务器，再从 FTP 服务器下载或上传文件。下面介绍 Python 编写 FTP 客户端程序的方法。

1. ftplib 模块

在 Python 系统默认安装的 ftplib 模块中，定义了 FTP 类。应用 ftplib 模块中的 FTP 类，可以方便地编写 FTP 客户端程序，用于上传或下载文件。

2. FTP 类的常用方法

FTP 类的常用方法如表 8.1 所示。

表 8.1　FTP 类的常用方法

方法	说明
ftp = FTP()	创建 FTP 对象
ftp.connect('IP', PORT)	连接 FTP 服务器，参数为服务器 IP 和端口
ftp.login('user', 'password')	登录用户名和密码
ftp.cmd('path')	进入远程目录
ftp.quit()	退出 FTP
ftp.dir()	显示目录下所有目录的信息
ftp.nlst()	获取目录下的文件
ftp.mkd(pathname)	新建远程目录
ftp.rmd(dirname)	删除远程目录
ftp.pwd()	返回当前所在位置
ftp.delete(filename)	删除远程文件
ftp.rename(fromname, toname)	将 fromname 改名为 toname
ftp.storbinaly('STOR filename', file_handel,bufsize)	上传目标文件
ftp.retrbinary('RETR filename, file_handel,bufsize)	下载 FTP 文件

3. 应用示例

【例 8-3】 设在 FTP 服务器（IP 为 129.168.1.1，端口号为 21）上有目录 test，该目录下有文件 hello.c。编写一个 FTP 客户端程序，将 FTP 服务器上的 hello.c 文件下载到客户机的 pytest 目录下。

程序代码如下：

```
from ftplib import FTP

ftp = FTP()                              # 创建FTP对象
timeout = 30                             # 设定传输超时的时间
port = 21                                # 端口号
ip = '192.168.1.1'                       # FTP服务器的IP地址
username = 'admin'                       # 登录用户名
passwd = '123456'                        # 用户密码
trFileName = 'hello.c'                   # 设置要传输的文件

ftp.connect(ip,port, timeout)            # 连接FTP服务器
ftp.login(username, passwd)              # 用户登录服务器
print(ftp.getwelcome())                  # 获得欢迎信息
```

```
ftp.cwd('/test')                          # 设置FTP路径（FTP服务器）
ftp.dir()                                 # 显示FTP路径目录下的文件

path = '/pytest/' + trFileName            # 文件保存路径（客户机）
f = open(path,'wb')                       # 打开要保存文件
filename = 'RETR ' + trFileName           # 保存FTP文件
ftp.retrbinary(filename, f.write)         # 保存FTP上的文件
print(filename)                           # 显示从FTP服务器下载的文件

ftp.quit()                                # 退出FTP服务器
```

在 FTP 服务器端运行 FTP 服务程序，然后在客户机端运行本程序。程序运行程序结果如下：

```
220 Welcome to Gxnn.com FTP Server!
-rwx------ 1 user group     251 Mar 13 13:28 hello.c
-rwx------ 1 user group   19521 Jul 24 22:23 hello.exe
-rwx------ 1 user group   4146 Jul 24 22:23 hello.o
RETR hello.c
```

这时，在客户机端的 pytest 目录下，可以看到从 FTP 服务器下载的 hello.c 文件。

8.3.2　基于 TCP 的端口扫描器

端口扫描是指通过 TCP 握手或别的方式判别一个给定主机上的某些端口是否处于开放或监听状态。下面编写一个简单的端口扫描器。

【例 8-4】 编写一个简易的端口扫描器程序。

```
from socket import *
import threading
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext                # 导入滚动文本框的模块

lock = threading.Lock()
openNum = 0
threads = []
host_ip = '192.168.1.1'
#str_port = ''

def portScanner(host,port):
    global openNum
    try:
        s = socket(AF_INET,SOCK_STREAM)         # 建立基于TCP的套接字对象
        s.connect((host,port))
        lock.acquire()                          # 多线程互斥锁
```

```
            openNum += 1                                # 统计打开的端口数
            str_port = '[+] %d open' % port
            txt.insert(tk.INSERT, str_port+'\n')        # 在文本框中显示扫描的结果
            lock.release()                              # 释放线程互斥锁
            s.close()
    except:
            pass

# 扫描按钮的方法
def scan():
    global openNum
    txt.insert(tk.INSERT,"正在扫描……\n")
    for port in range(1,500):
        setdefaulttimeout(1)
        t = threading.Thread(
            target=portScanner,
            args=(host_ip, port))
        threads.append(t)
        t.start()
    txt.insert(tk.INSERT,'[*] The scan is complete!\n')

#创建窗体
win = tk.Tk()                           #创建一个窗体对象
win.title("Python  端口扫描器")         # 设置窗体标题

# 创建一个标签框架容器
monty = tk.LabelFrame(win, text=" 扫描端口 ")
monty.grid(column=0, row=0, padx=10, pady=10)

# 按钮
scanBtn = ttk.Button(monty, text="扫描端口!", command=scan)
scanBtn.grid(column=2, row=1)

# 滚动文本框
scrolW = 50                             # 设置文本框的长度
scrolH = 15                             # 设置文本框的高度
txt = scrolledtext.ScrolledText(monty,
              width=scrolW, height=scrolH)
txt.grid(column=0, columnspan=3)        # columnspan 将三列合并成一列

win.mainloop()                          # 当调用mainloop()时,才会在窗口中显示
```

程序运行结果如图 8.6 所示。



图 8.6　简易端口扫描器

8.3.3　远程控制计算机

下面介绍在一台计算机上发送指令远程控制另一台计算机操作的示例。

该控件系统由服务器端程序和客户机程序组成，运行客户机程序的计算机发送操作指令，控制另一台运行服务器端程序的计算机。

【例 8-5】 编写远程控制计算机操作的程序。

（1）服务器端程序

程序代码如下：

```
from socket import *
import os
import sys

HOST = '127.0.0.1'
PORT = 4321
ADDR = (HOST, PORT)
ss = socket(AF_INET, SOCK_STREAM, 0)
ss.bind(ADDR)
ss.listen(10)
flag = True

while True:
    print('等待客户机连接……\n')
    cs, caddr = ss.accept()
    print('连接的客户机来自：', caddr)
    str = '欢迎你访问本服务器！'
    cs.sendall(bytes(str, 'UTF-8'))
    while True:
        msg = cs.recv(1024).decode()
        print('接收客户机信息：', msg)
        if(msg == "dir"):
            os.system('dir')
            break
```

创建和设置套接字对象（HOST 至 flag = True 各行）

接收远程命令信息（msg = cs.recv(1024).decode()）

执行“列文件目录”命令（if(msg == "dir") 至 break）

```
        if(msg == "shut"):
            os.system('shutdown -r -t 0')
            break
        if(msg == "quit"):
            cs.close
            sys.exit(0)
        cs.close()
ss.close()
```

执行“重启计算机”命令

执行“执行退出系统”命令

（2）客户机程序

程序代码如下：

```
from socket import *
import socket
import sys;
import tkinter
from tkinter import scrolledtext                    # 导入滚动文本框的模块

win = tkinter.Tk()
win.title("客户端程序")                              # 添加标题

# 创建一个容器
monty = tkinter.LabelFrame(win, text=" 发送指令信息 ") # 创建LabelFrame容器
monty.grid(column=0, row=0, padx=10, pady=10)   # padx,pady为容器外围空间

# 滚动文本框
scrolW = 60                                         # 设置文本框的长度
scrolH = 5                                          # 设置文本框的高度

txt_recv = scrolledtext.ScrolledText(monty, width=scrolW, height=scrolH)
txt_recv.grid(column=0, columnspan=4)               # columnspan将三列合并成一列

HOST = '127.0.0.1'
PORT = 4321
ADDR = (HOST, PORT)
global cs

def conn():
    global cs
    cs = socket.socket()
    cs.connect(ADDR)    # 向服务器发起连接请求
    data = cs.recv(1024).decode()

def com_dir():
    global cs
    conn()
    str = "dir"
    cs.sendall(bytes(str, 'UTF-8'))
    cs.close()
```

连接远程被控制的计算机

发送“列文件目录”的命令

```
    def com_shut():
        conn()
        str = "shut"
        cs.sendall(bytes(str, 'UTF-8'))
        cs.close()
```

发送“重启计算机”的命令

```
    def com_quit():
        global cs
        conn()
        str = "quit"
        cs.sendall(bytes(str, 'UTF-8'))
        cs.close()
```

发送“退出系统”的命令

```
# 按钮
action1 = tkinter.Button(monty, text="退出", command=exit)
action1.grid(column=0, row=1)

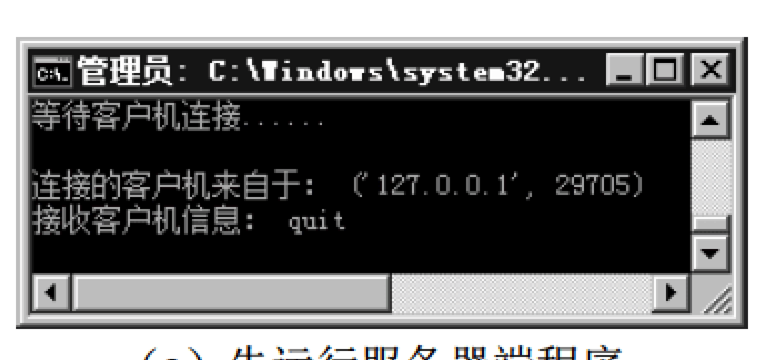
action2 = tkinter.Button(monty, text="列文件目录", command=com_dir)
action2.grid(column=1, row=1)

action3 = tkinter.Button(monty, text="服务器关机", command=com_shut)
action3.grid(column=2, row=1)

action3 = tkinter.Button(monty, text="关闭服务程序", command=com_quit)
action3.grid(column=3, row=1)

win.mainloop()
```

运行程序时，首先运行服务器端程序，然后再运行客户机端程序。程序运行结果如图 8.7 所示。



（a）先运行服务器端程序



（b）后运行客户机端程序

图 8.7　远程控制计算机操作

8.3.4　网络域名解析

在 Python 中，域名通常是一个字符串的形式。进行域名解析时，需要去掉字符串头尾的空格，这时需要用到 Python 的 strip()函数。

strip()函数的一般语法格式为：

```
str.strip([chars])
```

其中，chars 为移除字符串头尾指定的字符。

例如：

```
str = "*****this is string example...wow!!!*****"
print(str.strip('*'))
```

输出结果如下：

```
this is string example...wow!!!
```

当参数 chars 为空格时，strip()要写成无参函数。

当域名写成“http://xxxxx.xxxx.xxxxx”形式时，则需要使用字符串运算符[:]去除“http://”。

【例 8-6】 设文本文件 urllist.txt 中存放网站域名如下：

http://sdk.mobcent.com

http://www.baidu.com

编写一个域名解析程序，解析对应的 IP 地址，并保存到文件 iplist.txt 中。

程序代码如下：

```
import socket

def URL2IP():
   for oneurl in urllist.readlines():
       url = str(oneurl.strip())[7:]   ← 去除字符串中前 7 个字符“http://”
       print(url)
       try:
           ip = socket.gethostbyname(url)   ← 核心语句：从域名中解析 IP 地址
           print(ip)
           iplist.writelines(str(ip)+"\n")   ← 将解析的 IP 写入文件
       except:
           print("this URL 2 IP ERROR ")

try:
    urllist = open("D:\\urllist.txt","r")
    iplist = open("D:\\iplist.txt","w")
    URL2IP()
    urllist.close()
    iplist.close()
    print("complete !")
except:
    print("ERROR !")
```

运行程序，其结果如下：

```
sdk.mobcent.com
103.235.239.10
www.baidu.com
119.75.216.20
complete !
```

此时，新生成的文件 iplist.txt 中，保存了解析的 IP 地址，其内容如下：

```
103.235.239.10
119.75.216.20
```

8.4　网络爬虫实战入门



视频录像

8.4.1　抓取网页数据

网页数据抓取指从网络资源上抓取网页中的一些有用数据或网络文件数据。其基本过程是获取网络上的网页内容或文件，然后再进行正则匹配处理。

抓取网页数据需要用到 urllib.request 模块和 BeautifulSoup 模块，下面对这两个模块进行详细介绍。

1. urllib 库

urllib 是 Python 内置的标准库模块，使用它可以像访问本地文本文件一样读取网页的内容。Python 的 urllib 库模块包括以下 4 个模块：

- urllib.request 请求模块；
- urllib.error 异常处理模块；
- urllib.parse url 解析模块；
- urllib.robotparser 解析模块。

其中，urllib.request 请求模块主要用于打开和读取 URL 资源；urllib.parse 模块主要用于解析 URL 资源。下面的例子将结合使用这两个模块，说明 urllib 库模块的使用方法。

2. urllib.request 模块的常用方法

urllib.request 模块的常用方法如表 8.2 所示。

表 8.2　urllib.request 模块的常用方法

方法	说明
urllib.request.urlopen()	建立连接
urllib.request.install_opener(opener)	设置代理
urllib.request.build_opener()	处理连接
urllib.request.Request(url, data)	连接请求
urllib.request.urlretrieve(url, filename=None)	把网络对象复制到本地

下面通过示例说明 urllib.request 模块常用方法的使用。其基本步骤为：

（1）导入 urllib.request 模块

```
from urllib import request
```

（2）连接要访问的网站，发起请求

```
resp = request.urlopen("http://网站IP地址")
```

（3）获取网站代码信息

```
print(resp.read().decode())
```

【例 8-7】 应用 urllib.request.urlopen()方法连接网站，抓取页面代码。

程序代码如下：

```
import urllib.request

response=urllib.request.urlopen("http://www.baidu.com/")
print(response.info())
print('\n*****************************************************\n')
print(response.getcode())
print('\n*****************************************************\n')
print(response.read())
```

程序运行结果如图 8.8 所示。

```
Connection: Close
Vary: Accept-Encoding
Set-Cookie: BAIDUID=4D73A9AAB69C29336E75B2900F068882:FG=1; expires=Thu, 31-Dec-37
23:55:55 GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: BIDUPSID=4D73A9AAB69C29336E75B2900F068882; expires=Thu, 31-Dec-37 23:55:55
GMT; max-age=2147483647; path=/; domain=.baidu.com
Set-Cookie: PSTM=1520904857; expires=Thu, 31-Dec-37 23:55:55 GMT; max-age=2147483647;
path=/; domain=.baidu.com
Set-Cookie: BDSVRTM=0; path=/
Set-Cookie: BD_HOME=0; path=/
Set-Cookie: H_PS_PSSID=1440_21110_18559_17001_20927; path=/; domain=.baidu.com
P3P: CP=" OTI DSP COR IVA OUR IND COM "
Cache-Control: private
Cxy_all: baidu+befc65584d413c23251555985f4d7d52
Expires: Tue, 13 Mar 2018 01:33:30 GMT
X-Powered-By: HPHP
Server: BWS/1.1
X-UA-Compatible: IE=Edge,chrome=1
BDPAGETYPE: 1
BDQID: 0xf64759790004ca35
BDUSERID: 0
************************************************************
200
************************************************************
b'<!DOCTYPE html>\n<!--STATUS OK-->\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r
\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n
\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r
\n\r\n\r\n\r\n\t\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r
\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\t\r\n      \r\n\t
\t\t      \r\n\t\r\n\t\t\t      \r\n\t\r\n\t\t\t      \r\n\t\r\n\t\t\t      \r\n\t\t\t   \r\n
\r\n\t\r\n      \r\n\t\t\t      \r\n\t\r\n\t\t\t      \r\n\t\r\n\t\t\t      \r\n\t\r\n\t\t\t
   \r\n\t\t\t   \r\n\r\n\r\n\r\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\r\n\n<html>
\n<head>\n   \n   <meta http-equiv="content-type" content="text/html;charset=utf-8">\n   <meta
http-equiv="X-UA-Compatible" content="IE=Edge">\n\t<meta content="always" name="referrer">\n
 <meta name="theme-color" content="#2932e1">\n   <link rel="shortcut icon" href="/favicon.ico"
type="image/x-icon" />\n   <link rel="search" type="application/opensearchdescription+xml"
href="/content-search.xml" title="\xe7\x99\xbe\xe5\xba\xa6\xe6\x90\x9c\xe7\xb4\xa2" />
\n   <link rel="icon" sizes="any" mask
```

图 8.8　抓取的网页内容

3. BeautifulSoup 模块

BeautifulSoup 是 Python 的一个解析、遍历、维护网页文档“标签”的功能库模块，其主要功能是从连接的网站上通过解析文档抓取网页数据。BeautifulSoup 模块提供了一些功能函数用来处理导航、搜索、修改分析树等。

BeautifulSoup 模块使用时不需要考虑编码方式，自动将输入文档转换为 Unicode 编码，输出文档转换为 UTF-8 编码。

（1）安装 BeautifulSoup 模块

BeautifulSoup 模块不是 Pyton 系统自带模块，因此在使用前必须用 pip 安装该模块。用 pip 安装 BeautifulSoup 模块的命令如下：

```
pip install beautifulsoup4
```

安装结果如图 8.9 所示。

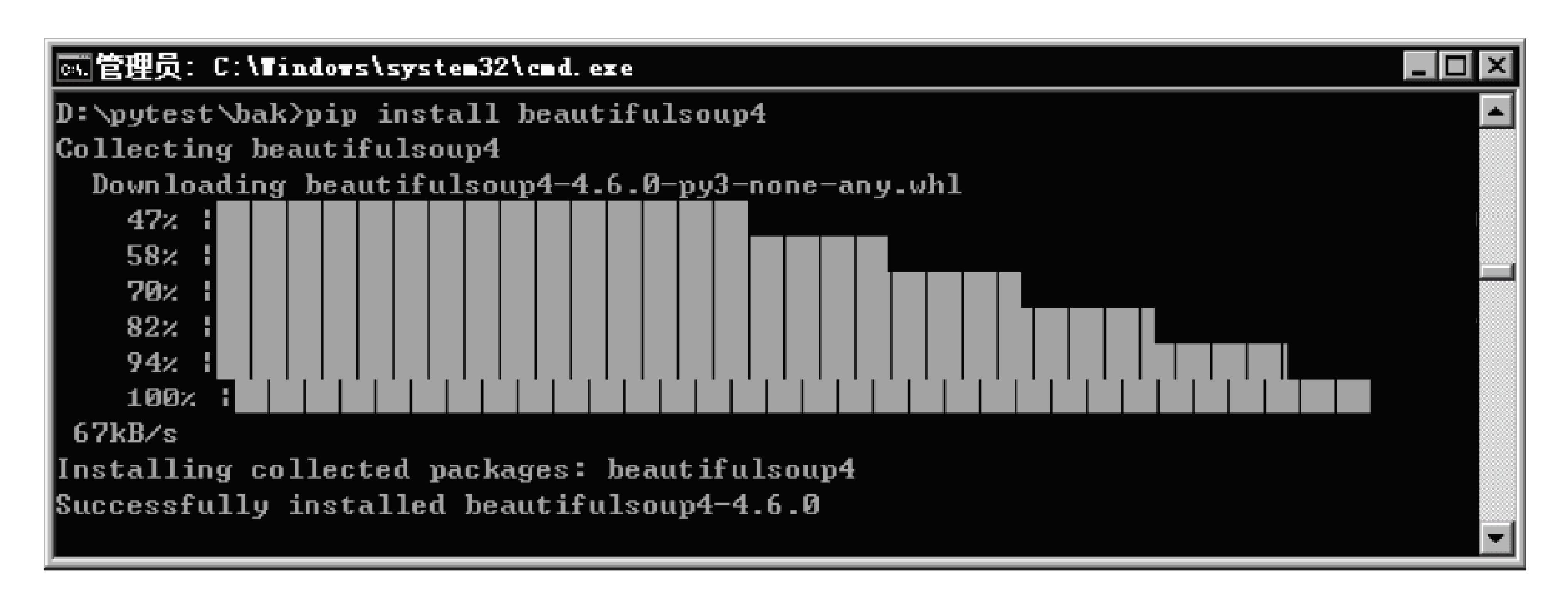


图 8.9 安装 BeautifulSoup 模块

（2）BeautifulSoup 模块的基本元素

BeautifulSoup 模块的基本元素如表 8.3 所示。

表 8.3 BeautifulSoup 模块的基本元素

基本元素	说明
Tag	标签，最基本的信息组织单元，分别用< >和</>标明开头和结尾
Name	标签的名字, … 的名字是 p，格式是<tag>.name
Attributes	标签的属性，字典形式组织，格式为<tag>.attrs
NavigableString	标签内非属性字符串，格式为<tag>.string
Comment	标签内字符串的注释部分，一种特殊的 Comment 类型

（3）标签树

在解析网页文档的过程中，需要应用 BeautifulSoup 模块对 HTML 内容进行遍历。

设有如下的一个 HTML 文档：

```
<html>
  <head>
    ⋮
  </head>
```

```
    <body>
      <p class="title"> The demo Python Project.</p>
      <p class="course"> Python is a programming language.
          <a href="http://www.icourse163.com"> Basic Python </a>
       <a href="http://www.python.org"> Advanced Python </a>
       </p>
     </body>
</html>
```

将其文档标签绘成树形结构，该结构称为“标签树”，如图 8.10 所示。

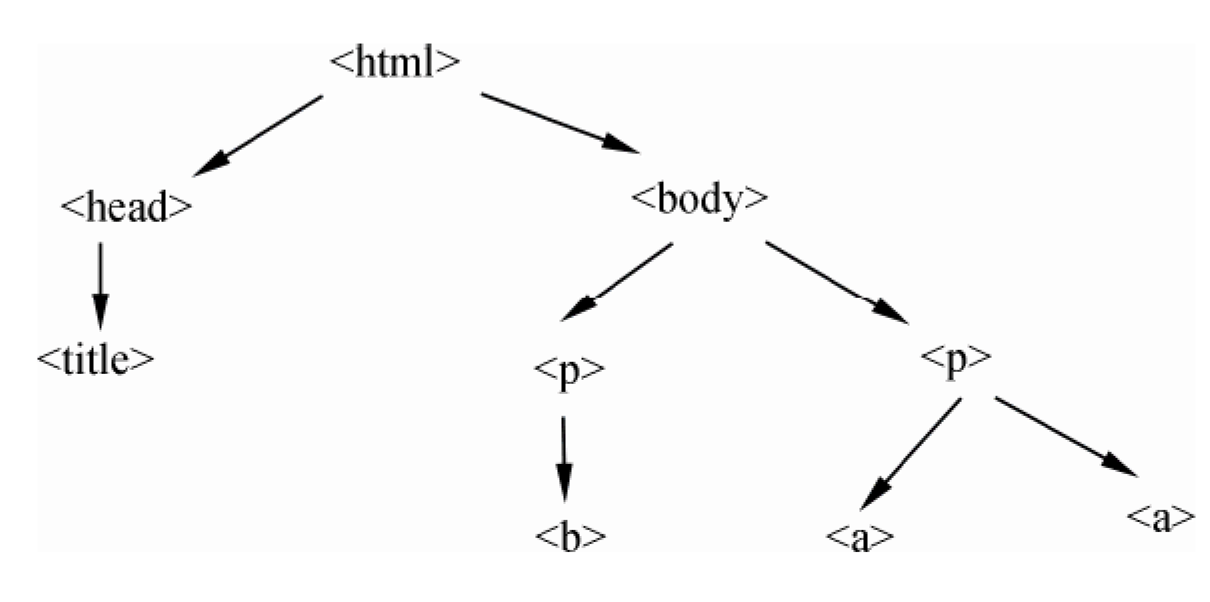


图 8.10　标签树

（4）BeautifulSoup 模块对象“标签树”的上行遍历属性

BeautifulSoup 模块对象“标签树”的上行遍历属性如表 8.4 所示。

表 8.4　“标签树”的上行遍历属性

属性	说明
.parent	节点的父标签
.parents	节点先辈标签的迭代类型，用于循环遍历先辈节点

（5）BeautifulSoup 模块对象“标签树”的下行遍历属性

BeautifulSoup 模块对象“标签树”的下行遍历属性如表 8.5 所示。

表 8.5　“标签树”的下行遍历属性

属性	说明
.contents	子节点的列表，将<tag>所有子节点存入列表中
.children	子节点的迭代类型，用于循环遍历子节点
.descendants	子孙节点的迭代类型，用于循环遍历所有子孙节点

（6）BeautifulSoup 模块对象的信息提取方法

设 BeautifulSoup 模块对象为 soup，则其常用信息提取方法如表 8.6 所示。

表 8.6　BeautifulSoup 模块对象的常用信息提取方法

属性	说明
soup.find_all()	搜索信息，返回一个列表类型，存储查找的结果
soup.find()	搜索且只返回一个结果信息

8.4.2 网络爬虫简介

1. 什么是网络爬虫

网络爬虫（又被称为网页蜘蛛，网络机器人），是一种按照一定的规则，自动抓取万维网信息的程序。

网络爬虫可以理解为在网络上爬行的一只蜘蛛，互联网就比作一张大网，而爬虫便是在这张网上爬来爬去的蜘蛛，如果它遇到资源，那么它就会抓取下来。让网络爬虫抓取什么内容，则由编写的程序控制。

2. 爬虫的基本流程

编写爬虫程序的基本流程如下：

（1）发起请求

通过 HTTP 库向目标站点发起请求，也就是发送一个 Request，请求可以包含额外的 header 等信息，等待服务器响应。

（2）获取响应内容

如果服务器能正常响应，会得到一个 Response，Response 的内容便是所要获取的页面内容，类型可能是 HTML、JSON 字符串、二进制数据（图片或视频）等类型。

（3）解析内容

得到的内容可能是 HTML，可以用正则表达式，页面解析库进行解析；可能是 JSON，可以直接转换为 JSON 对象解析；也可能是二进制数据，可以做保存或进一步的处理。

（4）保存数据

保存形式多样，可以存为文本，也可以保存到数据库，或保存特定格式的文件。

3. 举例

【例 8-8】 爬取最新电影的影评信息。

设有某影视网站（https://xxxx.xxxx.xxx），现介绍爬取其网站中的最新影评信息。

爬取网站信息的主要步骤如下：

（1）获取网站页面的 HTML 代码

下面代码段可以获取到网站页面的数据。

```
from urllib import request
resp = request.urlopen('https://xxxx.xxxx.xxx/xxxxxx/xxxxxx/')
html_data = resp.read().decode('utf-8')
```

其中，https:// xxxx.xxxx.xxx/xxxxxx /xxxxx 是一个推介最新上映电影的网站。

若在代码段后面添加语句：

```
print(html_data)
```

则可以显示电影网站页面的 HTML 代码，如图 8.11 所示。

（2）对得到的 HTML 代码进行解析，提取需要的数据

在 python 中使用 BeautifulSoup 库进行 HTML 代码的解析。BeautifulSoup 使用的格式如下：

```
<!DOCTYPE html>
<html lang="zh-cmn-Hans" class="">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <meta name="renderer" content="webkit">
    <meta name="referrer" content="always">
    <title>
    杭州 - 在线购票&amp;影讯
</title>

    <meta name="baidu-site-verification" content="cZdR4xxR7RxmM4zE" />
    <meta http-equiv="Pragma" content="no-cache">
    <meta http-equiv="Expires" content="Sun, 6 Mar 2005 01:00:00 GMT">
```

图 8.11　抓取电影网站页面的 HTML 代码

```
BeautifulSoup(html, "html.parser")
```

其中，第一个参数为需要提取数据的 HTML；第二个参数是指定解析器。使用 find_all()读取 HTML 标签中的内容。

但是 HTML 中有很多个标签，该读取哪些标签呢？最简单的办法是打开所爬取网页的 HTML 代码，然后查看需要的数据在哪个 HTML 标签中，再进行读取就可以了，如图 8.12 所示。

```
<div id="nowplaying">
    <div class="mod-hd">
        <h2>正在上映</h2>
    </div>
    <div class="mod-bd">
        <ul class="lists">
                <li
                    id="26363254"
                    class="list-item"
                    data-title="战狼2"
                    data-score="7.5"
                    data-star="40"
                    data-release="2017"
                    data-duration="123分钟"
                    data-region="中国大陆"
                    data-director="吴京"
                    data-actors="吴京 / 弗兰克·格里罗 / 吴刚"
                    data-category="nowplaying"
                    data-enough="True"
                    data-showed="True"
                    data-votecount="120308"
                    data-subject="26363254"
                >
```

图 8.12　读取抓取的页面代码

从图 8-12 中可以看到，影片《战狼 2》的电影名称、评分、主演等信息都存放在<li class = “list-item” >标签中，而<li>标签又存放在<div id= “nowplaying” >标签中。

查看影片信息的程序代码如下：

```
from bs4 import BeautifulSoup as bs

soup = bs(html_data, 'html.parser')
nowplaying_movie = soup.find_all('div', id='nowplaying')
nowplaying_movie_list = nowplaying_movie[0].find_all('li', class_='list-
item')
```

其中，nowplaying_movie_list 是一个列表，可以用 print(nowplaying_movie_list[0])查看里面

的内容，如图 8.13 所示。

```
<li
   class="list-item"
   id="6390825"
   data-actors="查德维克·博斯曼/露皮塔·尼永奥/迈克尔·B·乔丹"
   data-category="nowplaying"
   data-director="瑞恩·库格勒"
   data-duration="135分钟(中国大陆)"
   data-enough="True"
   data-region="美国"
   data-release="2018"
   data-score="6.7"
   data-showed="True"
   data-star="35"
   data-title="黑豹"
   data-votecount="93517"
   data-subject="6390825"
>
<ul class="">
<li class="poster">
  <a class="ticket-btn" data-psource="poster"
     href="https://movie.douban.com/subject/?from=playing_poster"
     target="_blank">
    <img alt="黑豹" class="" rel="nofollow"
     src="https://img3.doubanio.com/view/photo/public/p23434.jpg"/>
  </a>
</li>
```

图 8.13　查找网页代码中有用的属性

从图 8.13 中可以看到，标签<li>的 data-subject 属性里面存放了影片的 id 号码，而在<img>标签的 alt 属性里面存放了影片的名字。因此通过这两个属性能得到影片的 id 和名称。说明，打开电影短评的网页需要用到电影的 id，所以需要对它进行解析。

解析影片的 id 和名称的程序代码如下：

```
nowplaying_list = []
for item in nowplaying_movie_list:
        nowplaying_dict = {}
        nowplaying_dict['id'] = item['data-subject']
        for tag_img_item in item.find_all('img'):
            nowplaying_dict['name'] = tag_img_item['alt']
            nowplaying_list.append(nowplaying_dict)
```

其中，列表 nowplaying_list 中就存放了电影网站页面中所发布最新影片的 id 和名称，可以使用 print(nowplaying_list)进行查看，如图 8.14 所示。

```
[
{'id': '6390825', 'name': '黑豹'},
{'id': '26861685', 'name': '红海行动'},
{'id': '26393561', 'name': '小萝莉的猴神大叔'},
{'id': '30152451', 'name': '厉害了，我的国'},
{'id': '26698897', 'name': '唐人街探案2'},
{'id': '26972275', 'name':'恋爱回旋'},
{'id': '26611804', 'name': '三块广告牌'},
{'id': '26649604', 'name': '比得兔'},
{'id': '26603666', 'name': '妈妈咪鸭'},
{'id': '26575103', 'name':'捉妖记2'},
{'id': '27085923', 'name': '灵魂当铺之时间典当'},
{'id': '25829175', 'name': '西游记女儿国'},
{'id': '25899334', 'name': '飞鸟历险记'},
{'id': '26995719', 'name': '金钱世界'},
{'id': '27114417', 'name': '祖宗十九代'},
{'id': '27176717', 'name': '熊出没·变形记'},
{'id': '25856453', 'name': '闺蜜2'},
{'id':'26731790', 'name': '十七岁 与青春化敌为友'},
{'id': '3036465', 'name': '爱在记忆消逝前'},
{'id': '26836837', 'name': '宇宙有爱浪漫同游'}
]
```

图 8.14　解析电影网站所发布最新电影的 id 和名称

对照抓取的影片名称与该电影网站页面中显示的影片名称（如图 8.15 所示），可以看到其影片名称是一致的。



图 8.15　电影网站页面上的电影名称

（3）对页面数据进行解析并输出结果

下面进行对最新电影短评网址进行解析。例如，《黑豹》的短评网址如下：

```
https://xxxx.xxxx.xxx/xxxxx/6390825/comments?status=P
```

其中 6390825 就是影片的 id。

打开影片《黑豹》的短评页面的 HTML 代码，可以发现关于评论的数据是在<div>标签的 comment 属性下，如图 8.16 所示。

```
<div class="avatar">
     <a title="苍甜" >
     </a>
</div>
<div class="comment">
  <h3>
     <span class="comment-info">
       <a
       href="https://www.douban.com/people/3438507/"
       class="">苍甜</a>
       <span>看过</span>
       <span class="allstar40 rating" title="推
       荐"></span>
       <span class="comment-time " title="2018-02-17
       07:50:45"> 2018-02-17 </span>
     </span>
    </h3>
    <p class=""> 挺喜欢漫威电影讲段子说笑话的水平，但
    也希望漫威影业能够做出更大胆的突破和尝试。终于他们
    拍出了《黑豹》这部气质独特、成熟的作品。
    </p>
</div>
```

图 8.16　评论的数据是在<div>标签的 comment 属性下面

对图 8.16 中<div>标签内容进行解析，其代码如下：

```
requrl = 'https://xxxx.xxxx.xxx/xxxxx/' + \
          nowplaying_list[0]['id'] + \
```

```
            '/comments' + \
            '?' +'start=0' + \
            '&limit=20'
    resp = request.urlopen(requrl)
    html_data = resp.read().decode('utf-8')
    soup = bs(html_data, 'html.parser')
    comment_div_lits = soup.find_all('div', class_='comment')
```

此时，在 comment_div_lits 列表中存放的就是<div>标签和 comment 属性下面的 HTML 代码了。

对 comment_div_lits 代码中的 HTML 代码继续进行解析，程序代码如下：

```
eachCommentList = [];
for item in comment_div_lits:
        if item.find_all('p')[0].string is not None:
            eachCommentList.append(item.find_all('p')[0].string)
```

使用 print(eachCommentList)查看 eachCommentList 列表中的内容，可以看到里面存着想要的影评。

（4）完整的程序代码

程序代码如下：

```
from urllib import request
from bs4 import BeautifulSoup as bs

# 分析网页函数
def getNowPlayingMovie_list():
    resp = request.urlopen('https://xxxx.xxxx.xxx/xxxxxx/xxxxxx/')
    html_data = resp.read().decode('utf-8')
    soup = bs(html_data, 'html.parser')
    nowplaying_movie = soup.find_all('div', id='nowplaying')
    nowplaying_movie_list = nowplaying_movie[0].find_all('li',
                                 class_='list-item')
    nowplaying_list = []
    for item in nowplaying_movie_list:
        nowplaying_dict = {}
        nowplaying_dict['id'] = item['data-subject']
        for tag_img_item in item.find_all('img'):
            nowplaying_dict['name'] = tag_img_item['alt']
            nowplaying_list.append(nowplaying_dict)
    return nowplaying_list

# 爬取评论函数
def getCommentsById(movieId, pageNum):
```

```
    eachCommentList = [];
    if pageNum>0:
         start = (pageNum-1) * 20
    else:
        return False
    requrl = 'https:/xxxx.xxxx.xxx/xxxxx/' + \
            movieId + '/comments' + \
            '?' +'start=' + str(start) + '&limit=20'
    #print(requrl)
    resp = request.urlopen(requrl)
    html_data = resp.read().decode('utf-8')
    soup = bs(html_data, 'html.parser')
    comment_div_lits = soup.find_all('div', class_='comment')
    for item in comment_div_lits:
        if item.find_all('p')[0].string is not None:
            eachCommentList.append(item.find_all('p')[0].string)
    return eachCommentList

def main():
    #循环获取第一个电影的前3页评论
    commentList = []
    NowPlayingMovie_list = getNowPlayingMovie_list()
    for i in range(3):
        num = i + 1
        commentList_temp = getCommentsById(NowPlayingMovie_list[0]['id'], num)
        commentList.append(commentList_temp)

    # 将列表中的数据转换为字符串
    comments = ''
    for k in range(len(commentList)):
        comments = comments + (str(commentList[k])).strip()
    print(comments)

# 主函数
main()
```

运行程序后，抓取到的影评结果如图 8.17 所示。

```
['挺喜欢漫威电影讲段子说笑话的水平，但也希望漫威影业能够做
出更大胆的突破和尝试。终于他们拍出了《黑豹》这部气质独特、
成熟的作品。\n ',
' 节奏实在不行，釜山街头大战之后就没戏了，可不是来看讲家族
纠葛史、一副苦大仇深的情感片的啊。\n ',
```

图 8.17　抓取到的影评

8.5　网络爬虫案例精选

8.5.1　爬取某网站大学排名榜

视频录像

【例 8-9】　爬取某网站大学排名前 20 名学校。

主要步骤如下：

（1）获取网站页面，分析代码结构特征。

（2）处理页面，提取相关信息。

（3）解析数据，输出结果。

编写程序时，定义三个函数，对应以上三个步骤。

首先分析网站的代码结构特征。从网站的代码可以看到，所有有用数据均从标签<tbody>开始，每一个排名数据都在<td></td>的标签中，如图 8-18 所示。

```
⊟ <tbody class="hidden_zhpm" style="text-align:center;">
⊟     <tr class="alt"><td>1</td>
            <td><div align="left">清华大学</div></td>
            <td>北京市</td>
            <td>95.9</td>
            <td><div align="left">北京大学</div></td>
            <td>北京市</td>
            <td>82.6</td>
            <td><div align="left">浙江大学</div></td>
            <td>浙江省</td>
            <td>80</td>
```

图 8.18　某大学排名网站的代码结构

程序代码如下：

```
import bs4
from urllib import request
from bs4 import BeautifulSoup

'''（1）获取网站页面'''
def getHTMLText(url):
    try:
        resp = request.urlopen(url)
        html_data = resp.read().decode('utf-8')
        return html_data
    except:
        return ""

'''（2）处理页面，提取相关信息'''
def fillUnivList(ulist, html):

    soup = BeautifulSoup(html, "html.parser")
    for tr in soup.find('tbody').children:    # 搜索'tbody'后面的子节点
        if isinstance(tr, bs4.element.Tag):
```

```
            tds = tr('td')
            ulist.append([tds[0].string, tds[1].string, tds[3].string])

'''（3）解析数据，格式化输出结果'''
def printUnivList(ulist, num):
    tplt = "{0:^10}\t{1:{3}^10}\t{2:^10}"
    print(tplt.format("排名", "学校名称", "总分", chr(12288)))
    for i in range(num):
        u = ulist[i]
        print(tplt.format(u[0], u[1], u[2], chr(12288)))

if __name__ == '__main__':
    uinfo = []
    url = 'http://xxx.xxxxx.xx/xxxxxx.html'
    html = getHTMLText(url)
    fillUnivList(uinfo, html)
    printUnivList(uinfo, 20)     #  输出前20个大学排名
```

程序运行结果如下：

```
排名            学校名称          总分
1               清华大学          95.9
2               北京大学          82.6
⋮
```

8.5.2 爬取网络版小说——《红楼梦》

1. 爬虫网络版小说《红楼梦》

打开《红楼梦》小说的目录页面（http://www.136book.com/hongloumeng/），如图 8.19 所示。

图 8.19 《红楼梦》小说目录页

《红楼梦》网站，找到每个目录对应的 URL，爬取其中的正文内容，然后保存到本地文件中。

2. 网页结构分析

在 IE 浏览器中打开小说《红楼梦》的目录页页面，按 F12 键打开审查元素菜单。可以看到网页前端的代码内容，如图 8.20 所示。

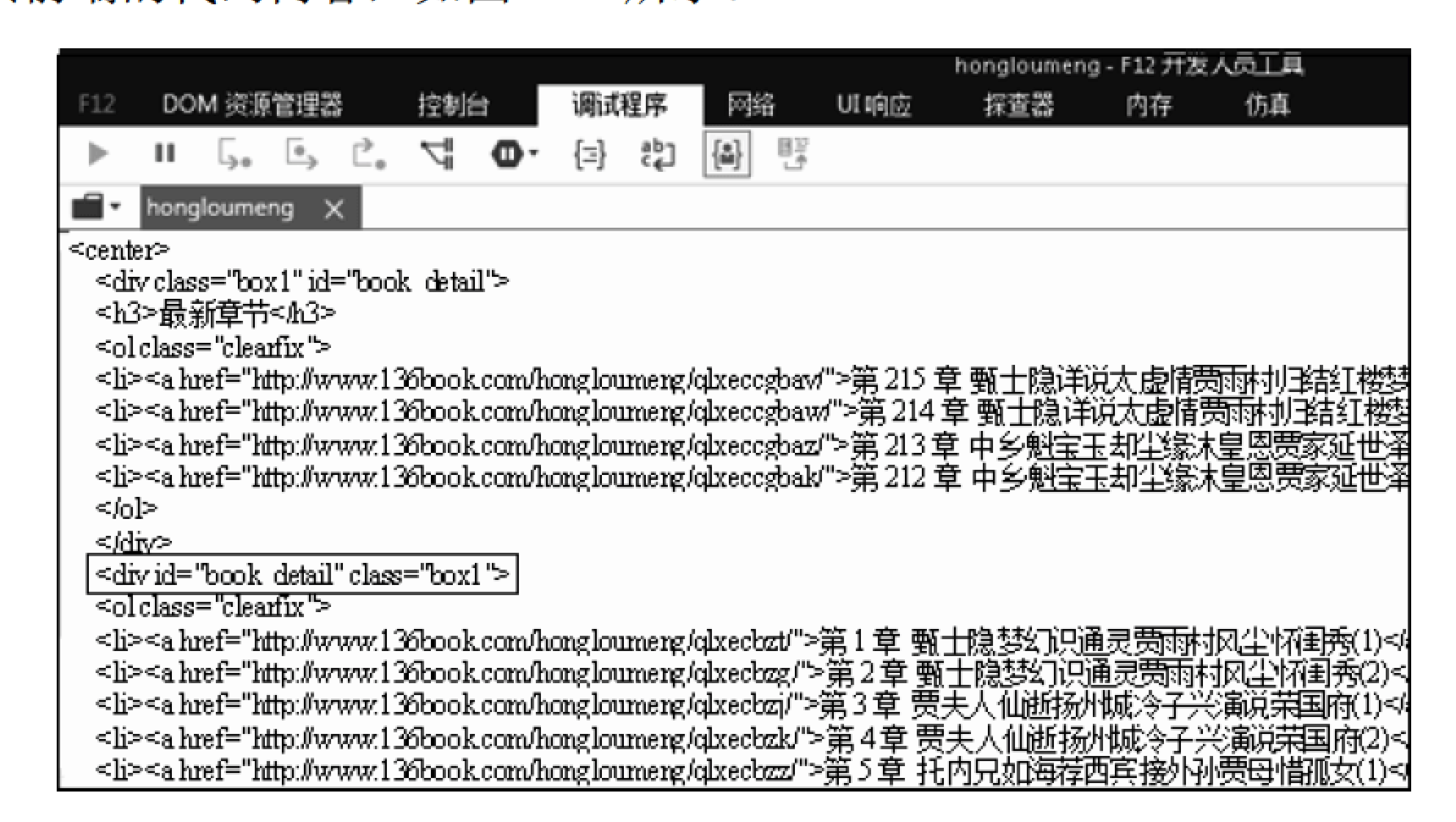


图 8.20　网页的代码

从图 8.20 中可以看到，每一章的链接地址都是有规则地存放在<li>标签中。而这些<li>标签又放在<div id="book_detail" class="box1">中。

3. 解析目录页

从目录页的代码结构，可以看到，所有的章节目录都放在<div id="book_detail" class="box1">的节点标签中。

【例 8-10】 抓取标签<div id="book_detail" class="box1">节点中的章节目录内容。

程序代码如下：

```
from urllib import request
from bs4 import BeautifulSoup

if __name__ == '__main__':
    # 目录页
    url = 'http://www.136book.com/hongloumeng/'
    head = {}
    req = request.Request(url, headers = head)
    response = request.urlopen(req)
    html = response.read()
    # 解析目录页
    soup = BeautifulSoup(html, 'lxml')
    # find_next找到第二个<div>
    soup_texts = soup.find('div', id = 'book_detail',
                                class_='box1').find_next('div')
    # 遍历ol的子节点，打印章节标题和对应的链接地址
```

```
    for link in soup_texts.ol.children:
        if link != '\n':
            print(link.text + ':  ', link.a.get('href'))
```

程序运行结果如图 8.21 所示。

```
第1章 甄士隐梦幻识通灵贾雨村风尘怀闺秀(1):   http://www.136book.com/hongloumeng/qlxecbzt/
第2章 甄士隐梦幻识通灵贾雨村风尘怀闺秀(2):   http://www.136book.com/hongloumeng/qlxecbzgba/
第3章 贾夫人仙逝扬州城冷子兴演说荣国府(1):   http://www.136book.com/hongloumeng/qlxecbzj/
第4章 贾夫人仙逝扬州城冷子兴演说荣国府(2):   http://www.136book.com/hongloumeng/qlxecbzk/
第5章 托内兄如海荐西宾接外孙贾母惜孤女(1):   http://www.136book.com/hongloumeng/qlxecbzz/
第6章 托内兄如海荐西宾接外孙贾母惜孤女(2):   http://www.136book.com/hongloumeng/qlxecbzw/
第7章 薄命女偏逢薄命郎葫芦僧乱判葫芦案(1):   http://www.136book.com/hongloumeng/qlxecbzv/
第8章 薄命女偏逢薄命郎葫芦僧乱判葫芦案(2):   http://www.136book.com/hongloumeng/qlxecbza/
第9章 游幻境指迷十二钗饮仙醪曲演红楼梦(1):   http://www.136book.com/hongloumeng/qlxecbzb/
第10章 游幻境指迷十二钗饮仙醪曲演红楼梦(2):   http://www.136book.com/hongloumeng/qlxecbzc/
第11章 贾宝玉初试云雨情刘姥姥一进荣国府(1):   http://www.136book.com/hongloumeng/qlxecbzd/
第12章 贾宝玉初试云雨情刘姥姥一进荣国府(2):   http://www.136book.com/hongloumeng/qlxecbze/
第13章 送宫花贾琏戏熙凤宴宁府宝玉会秦钟(1):   http://www.136book.com/hongloumeng/qlxecbzf/
```

图 8.21　章节目录

4. 单章节爬虫

刚才已经分析过网页结构。可以直接在浏览器中打开对应章节的链接地址，然后将文本内容提取出来，如图 8.22 所示。

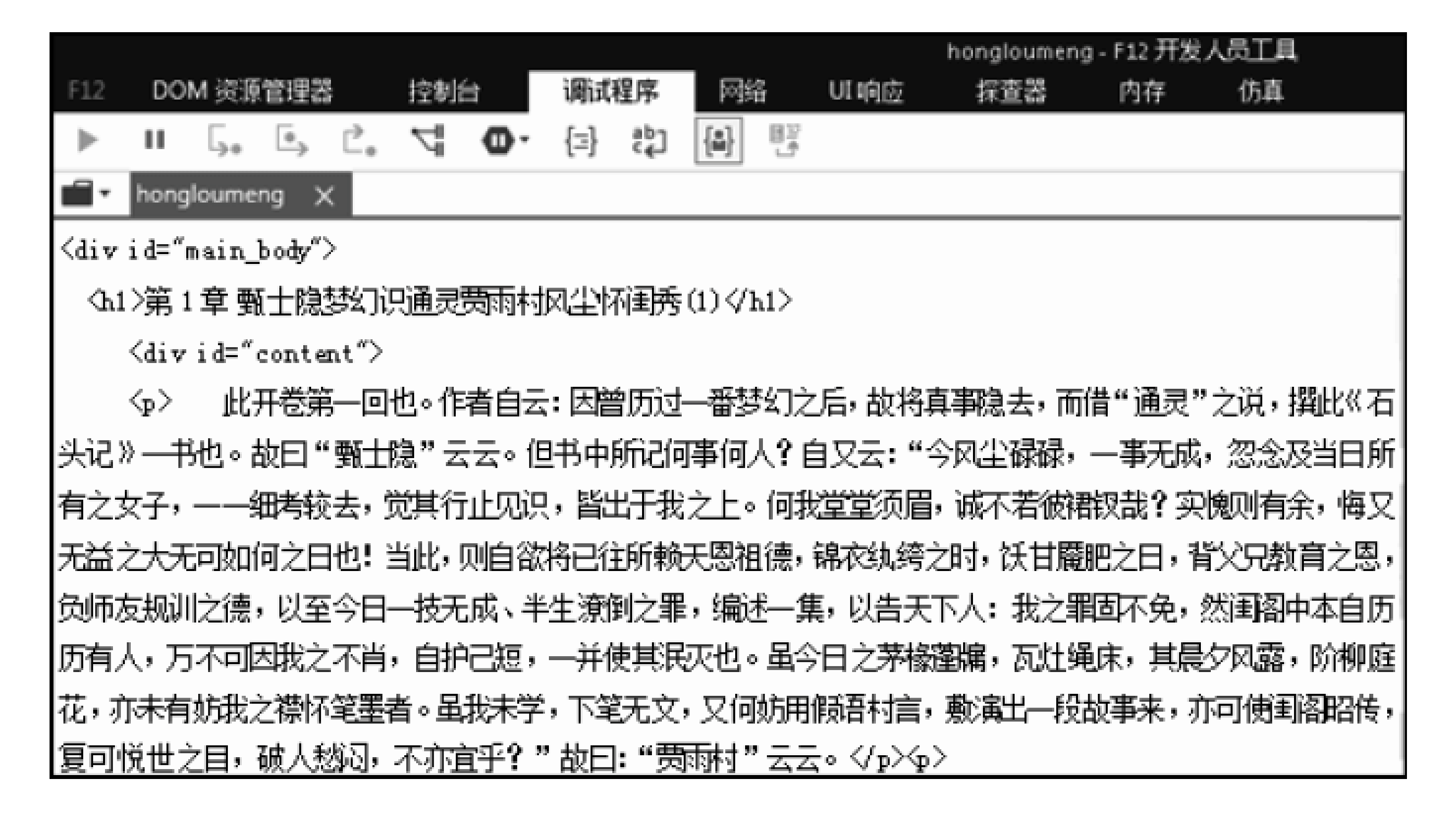


图 8.22　单章节的内容代码

从图 8.22 中可以看到，要爬取的内容全都包含在<div id="content">里面。

【例 8-11】 抓取标签<div id="content">中的单章节小说内容。

程序代码如下：

```
from urllib import request
from bs4 import BeautifulSoup

if __name__ == '__main__':
    # 第1章的网址
    url = 'http://www.136book.com/hongloumeng/qlxecbzt/'
    head = {}
    req = request.Request(url, headers = head)
    response = request.urlopen(req)
    html = response.read()
    # 创建request对象
```

```
    soup = BeautifulSoup(html, 'lxml')
    # 找出div中的内容
    soup_text = soup.find('div', id = 'content')
    # 输出其中的文本
    print(soup_text.text)
```

程序运行结果如图 8.23 所示。

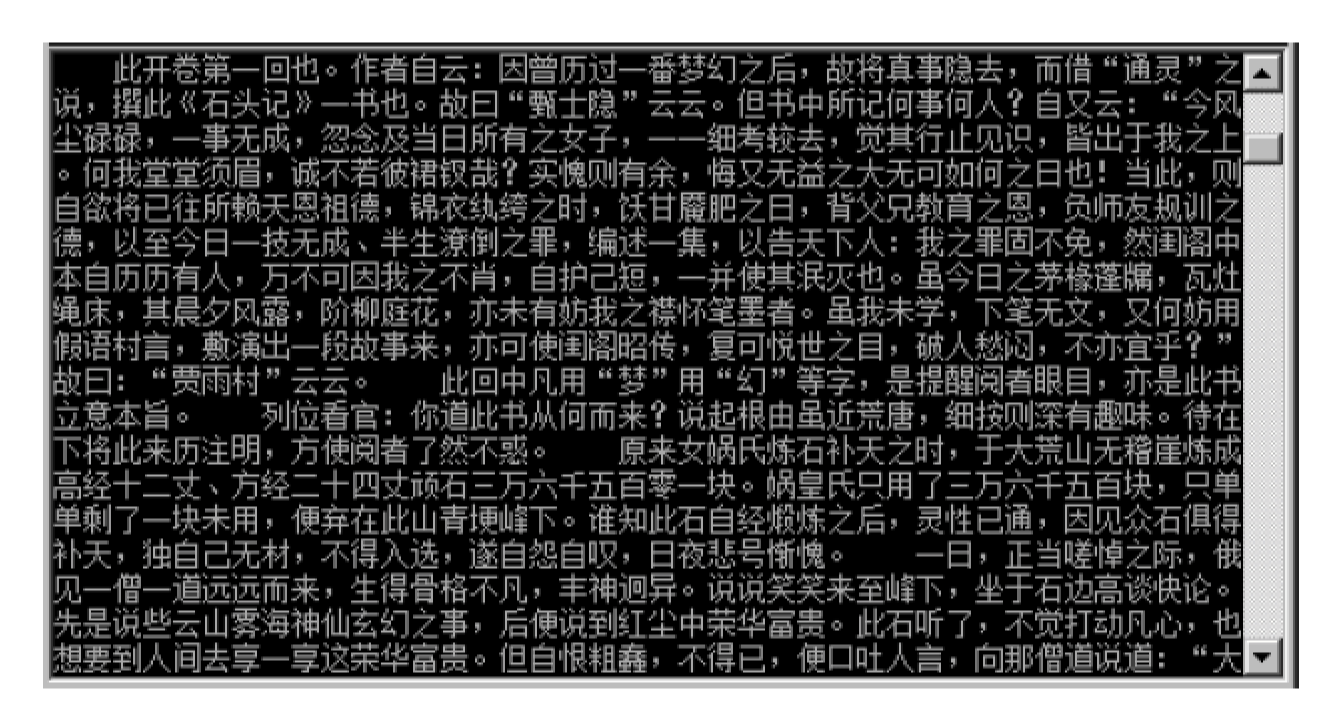


图 8.23　抓取到单章节的内容

5. 爬取全集内容

将每个解析出来的各章节链接循环代入到 url 中解析出来，并将其中的文本爬取出来，并且保存到本地 hongloumeng.txt 文件中。

【例 8-12】 解析《红楼梦》全集。

程序代码如下：

```
from urllib import request
from bs4 import BeautifulSoup

if __name__ == '__main__':
    url = 'http://www.136book.com/hongloumeng/'
    head = {}
    req = request.Request(url, headers = head)
    response = request.urlopen(req)
    html = response.read()
    soup = BeautifulSoup(html, 'lxml')
    soup_texts = soup.find('div', id = 'book_detail',
                           class_= 'box1').find_next('div')
    # 打开文件
    f = open('D:\hongloumeng.txt','w')
    # 循环解析链接地址
    for link in soup_texts.ol.children:
        if link != '\n':
            download_url = link.a.get('href')
            download_req = request.Request(download_url, headers = head)
            download_response = request.urlopen(download_req)
            download_html = download_response.read()
```

```
        download_soup = BeautifulSoup(download_html, 'lxml')
        download_soup_texts = download_soup.find('div', id = 'content')
        # 抓取其中文本
        download_soup_texts = download_soup_texts.text
        # 写入章节标题
        f.write(link.text + '\n\n')
        # 写入章节内容
        f.write(download_soup_texts)
        f.write('\n\n')
    f.close()
```

运行程序后，打开保存下载到本地的网络版小说文件 D:\hongloumeng.txt，可以看到抓取并解析的《红楼梦》全部内容，如图 8.24 所示。

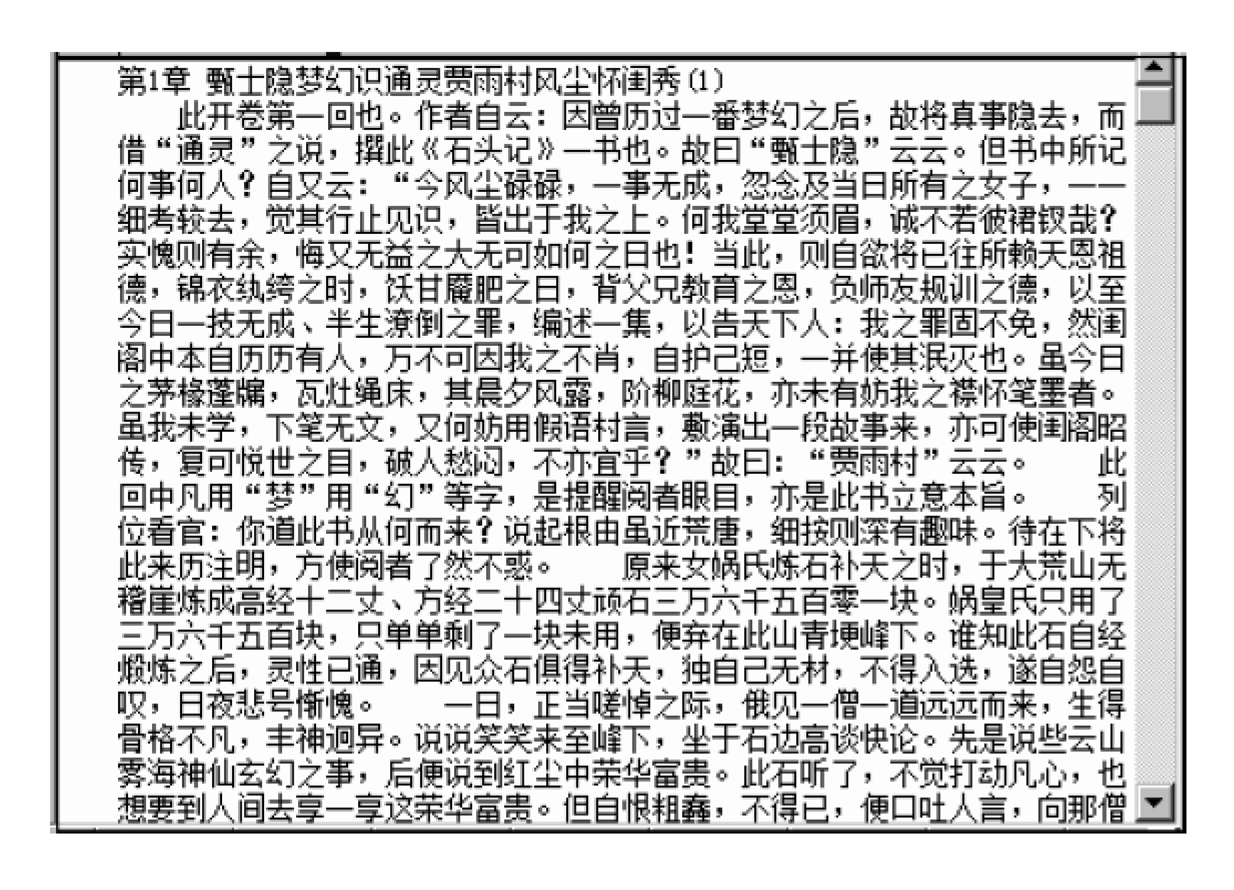

第1章 甄士隐梦幻识通灵贾雨村风尘怀闺秀(1)
此开卷第一回也。作者自云：因曾历过一番梦幻之后，故将真事隐去，而借“通灵”之说，撰此《石头记》一书也。故曰“甄士隐”云云。但书中所记何事何人？自又云：“今风尘碌碌，一事无成，忽念及当日所有之女子，一一细考较去，觉其行止见识，皆出于我之上。何我堂堂须眉，诚不若彼裙钗哉？实愧则有余，悔又无益之大无可如何之日也！当此，则自欲将已往所赖天恩祖德，锦衣纨绔之时，饫甘餍肥之日，背父兄教育之恩，负师友规训之德，以至今日一技无成、半生潦倒之罪，编述一集，以告天下人：我之罪固不免，然闺阁中本自历历有人，万不可因我之不肖，自护己短，一并使其泯灭也。虽今日之茅椽蓬牖，瓦灶绳床，其晨夕风露，阶柳庭花，亦未有妨我之襟怀笔墨者。虽我未学，下笔无文，又何妨用假语村言，敷演出一段故事来，亦可使闺阁昭传，复可悦世之目，破人愁闷，不亦宜乎？”故曰：“贾雨村”云云。 此回中凡用“梦”用“幻”等字，是提醒阅者眼目，亦是此书立意本旨。 列位看官：你道此书从何而来？说起根由虽近荒唐，细按则深有趣味。待在下将此来历注明，方使阅者了然不惑。 原来女娲氏炼石补天之时，于大荒山无稽崖炼成高经十二丈、方经二十四丈顽石三万六千五百零一块。娲皇氏只用了三万六千五百块，只单单剩了一块未用，便弃在此山青埂峰下。谁知此石自经煅炼之后，灵性已通，因见众石俱得补天，独自己无材，不得入选，遂自怨自叹，日夜悲号惭愧。 一日，正当嗟悼之际，俄见一僧一道远远而来，生得骨格不凡，丰神迥异。说说笑笑来至峰下，坐于石边高谈快论。先是说些云山雾海神仙玄幻之事，后便说到红尘中荣华富贵。此石听了，不觉打动凡心，也想要到人间去享一享这荣华富贵。但自恨粗蠢，不得已，便口吐人言，向那僧

图 8.24 抓取并解析的《红楼梦》全部内容

8.5.3 爬取天气预报信息

1. 分析天气预报网页的结构

要进行一个网络爬虫程序设计，首先要分析网页的代码结构。打开要获取天气预报信息的网站，进入北京地区天气的页面。网站页面如图 8.25 所示。



图 8.25 天气预报数据显示页面

打开网页的代码查看器（IE 浏览器按下 F12 按键），则可以看到当前页面的代码结构，如图 8.26 所示。

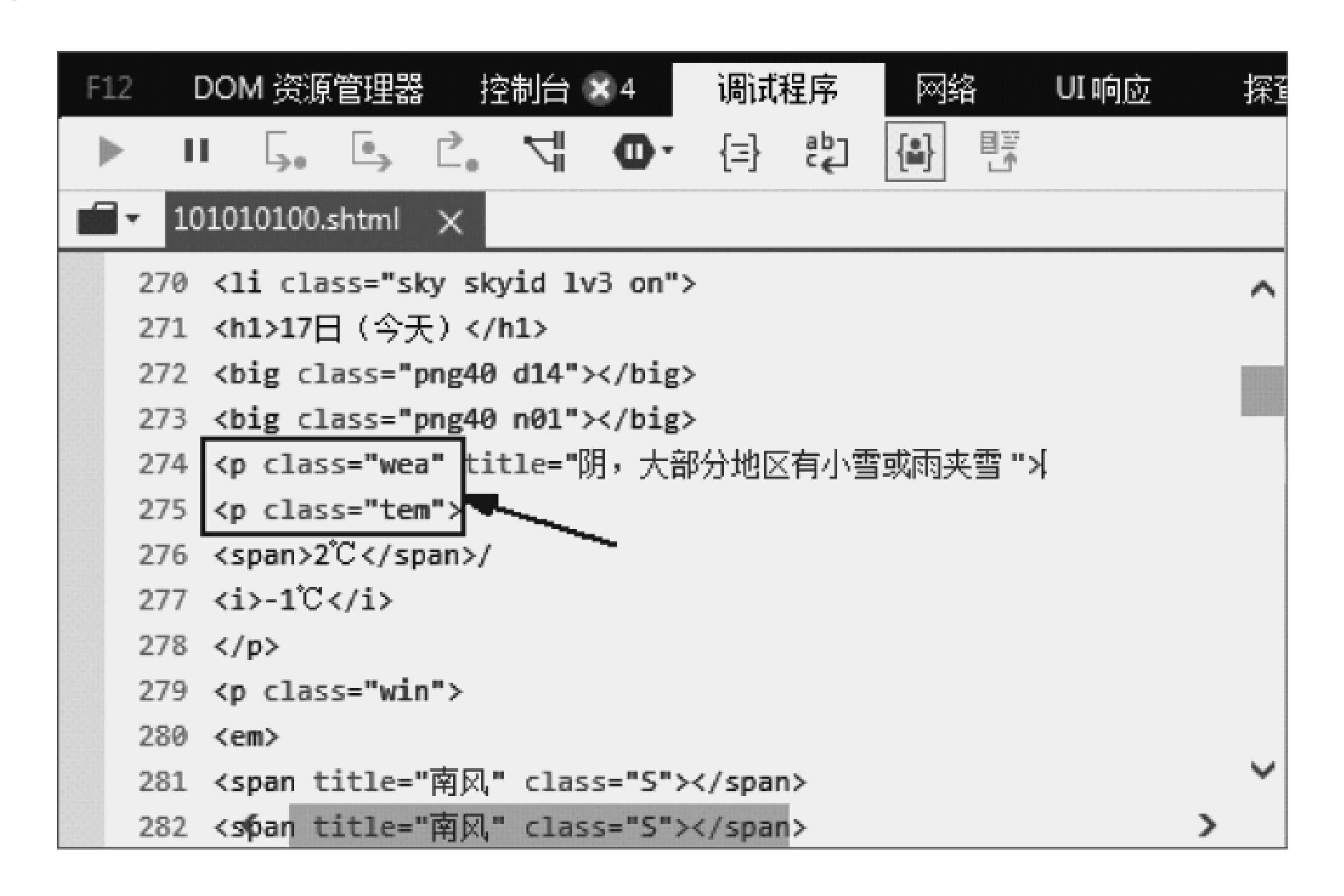


图 8.26　天气预报网页的代码结构

从图 8.26 中可以看到，当天的天气预报信息存放在<p class= “wea” >标签中，当天的气温存放在<p class= “tem” >标签中，其中最高气温存放在<span>标签中,最低气温存放在<i>标签中。

2. 解析网页数据

根据天气预报页面的代码结构分析，找到<p class= “wea” >标签和<p class= “tem” >标签，就能获得当天的天气信息和气温信息。

【例 8-13】 抓取标签<p class= “wea” >和标签<p class= “tem” >节点中的天气信息和气温信息。

程序代码如下：

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re, os

resp=urlopen('http://xxx.xxxx.xxx.xxx/101010100.shtml')
soup=BeautifulSoup(resp,'html.parser')

# 解析当天气温数据信息
tagToday=soup.find('p',class_="tem")

try:
    temperatureHigh = tagToday.span.string
except AttributeError as e:
    temperatureHigh =\
    tagToday.find_next('p',class_="tem").span.string
```

urlopen()方法

第一个包含 class="tem"的 p 标签即为存放今天天气数据的标签

有时没有最高温度，则用第 2 天的最高温度代替

获取第 2 天最高温度

```
temperatureLow = tagToday.i.string              # 解析当天最低温度
weather=soup.find('p',class_="wea").string      # 解析当天天气信息

print('最低温度:' + temperatureLow)
print('最高温度:' + temperatureHigh)
print('天气:' + weather)
```

程序运行结果如下：

```
最低温度:-1℃
最高温度:2℃
天气:阴，大部分地区有小雪或雨夹雪
```

8.5.4 网络爬虫利器——Requests

在前面介绍了爬取网络数据所使用的工具，主要是使用 Python 系统自带的 urllib 库模块。但 urllib 库模块有时用起来不方便，下面介绍编写网络爬虫程序的利器——Requests 模块。

1. 安装和导入 Requests 模块

可以使用 pip 命令来安装 Requests 模块：

```
pip install requests
```

安装好 Requests 模块后，就可以使用导入模块语句将其导入到程序中：

```
import requests
```

2. get 请求

Requests 模块的主要方法是 requests.get()，该方法用于向目标网站发起请求。其返回值的内容即为目标网站页面的 HTML 代码。

3. 应用示例

下面通过两个示例，说明 Requests 模块的应用方法。

【例 8-14】 修改例 8-13，应用 Requests 模块爬取天气信息。

```
import requests as req  <--  导入 Requests 模块
from bs4 import BeautifulSoup
import re, os

resp = req.get('http://xxx.xxx.xxx.xxx/101010100.shtml')  <--  get()方法
resp.encoding = "utf-8"
soup = BeautifulSoup(resp.text,'html.parser')
tagToday = soup.find('p',class_="tem")  <--  第一个包含 class="tem"的 p 标签即为存放今天天气数据的标签

try:
    temperatureHigh = tagToday.span.string  <--  有时候最高温度不显示，则用第2天的最高温度代替
except AttributeError as e:
```

```
temperatureHigh = \
tagToday.find_next('p',class_="tem").span.string    ← 获取第 2 天的最高温度

temperatureLow = tagToday.i.string                 # 获取最低温度
weather = soup.find('p',class_="wea").string       # 获取天气

print('最低温度:' + temperatureLow)
print('最高温度:' + temperatureHigh)
print('天气:' + weather)
```

【例 8-15】 爬取某图片网站上的图片，下载到本地计算机。

（1）分析图片网站页面的结构

进入图片网站，可以看到其图片页面，如图 8.27 所示。



图 8.27　图片网站显示的图片页面

（2）页面结构分析

在 IE 浏览器中按下 F12 键，打开代码调试器，可以看到网页前端的代码内容，如图 8.28 所示。



图 8.28　图片页面的代码内容

从图 8.28 中可以看到，每一幅图片的文件名、图片名称都有规则地存放在“img src=”项和“title=”项的后面。

（3）解析图片文件

通过分析图片页面的代码结构可以看到，每一幅图片的文件名、图片名称都有规则地存放在“img src=”项和“title=”项的后面，只要找到这些特征项，就可以把所有图片的文件名和图片名称解析出来。

使用 BeautifulSoup 模块的 select()方法，可以很方便地解析出图片信息：

```
Name = BeautifulSoup.select('h1')[0].string
Image = BeautifulSoup.select('img[title=\"' + Name + '\"]')
```

（4）程序

程序代码如下：

```
import requests
from bs4 import BeautifulSoup
import os

def Get_image_url(url):
    #传入页面的URL，得到所有图片所在的标签和图册的名字，并返回
    Res = requests.get(url)
    Soup = BeautifulSoup(Res.text,'lxml')

    Name = Soup.select('h1')[0].string
    Tag = 'img[title=\"' + Name + '\"]'
    Image = Soup.select(Tag)

    return Image,Name

def Download_Image(Image_url):
    #传入图片的URL，将图片保存在本地
    Image = requests.get(Image_url,stream=True)
    #将链接的最后一个字符串最为图片的名字
    name = Image_url.split('/')[-1]
    #保存图片
    with open(name,'wb') as f:
        f.write(Image.content)

def main():
    # 主函数
    url = "http://xxx.xxx.xxx/wallpaper_detail_54520.html"
    [Image,Name] = Get_image_url(url)
    # print(Name,Image)
    # 保存当前目录
```

```
    path = os.getcwd()
    # 创建保存图片的目录
    os.mkdir(Name)
    os.chdir(path + '\\' + Name)
    for I in Image:
        Download_Image(I['src'])
    # 返回之前的目录
    os.chdir(path)

if __name__ == '__main__':
    main()
```

运行程序后，在本地计算机的当前文件夹下，新生成一个名为“好看的动漫场景唯美意境图片桌面壁纸高清”的子文件夹，里面下载了一幅爬取到的图片。

8.5.5 爬取购物网站商品信息

1. 分析购物网站商品信息网页的结构

（1）购物网站商品信息网页页面

打开某购物平台，在站内搜索栏中输入“Python 爬虫”，则可以看到列出了所有名称中带有“Python 爬虫”书籍的商品信息，如图 8.29 所示。



图 8.29 显示名称中带有“Python 爬虫”书籍的商品信息的页面

（2）网页结构分析

在 IE 浏览器中按下 F12 键，打开代码调试器，可以看到网页前端的代码内容，如图 8.30 所示。



图 8.30　网页的代码分析

从图 8.30 中可以看到，每一种商品的名称、价格都是有规则地存放在“raw_title”项和“view_price”项的后面。

2. 解析商品信息

通过分析商品页面的代码结构可以看到，每一种商品的名称、价格都是有规则地存放在“raw_title”项和“view_price”项的后面，只要找到这些特征项，就可以把所有商品的名称和价格解析出来。

使用 re 模块的 findall()方法，可以很方便地应用正则表达式解析商品信息。

```
plt = re.findall(r'\"view_price\"\:\"[\d\.]*\"', html)    # 商品价格
tlt = re.findall(r'\"raw_title\"\:\".*?\"', html)         # 商品名称
```

【例 8-16】 编写程序，在购物网站爬取到有关“Python 爬虫”的商品信息。

程序代码如下：

```
import requests as req                    # 导入 Requests 模块
import re

def getHTMLText(url):
 try:
  r = req.get(url, timeout=30)            # 发送get()请求，返回Response对象
  r.raise_for_status()                    # 响应状态，若错误则，抛出异常
  r.encoding = r.apparent_encoding
  return r.text
 except:
  return ""

def parasePage(ilt, html):
 try:
  plt = re.findall(r'\"view_price\"\:\"[\d\.]*\"', html)
```

```
        tlt = re.findall(r'\"raw_title\"\:\".*?\"', html)
        for i in range(len(plt)):
            price = eval(plt[i].split(':')[1])
            title = eval(tlt[i].split(':')[1])
            ilt.append([price, title])
    except:
        print("")

def printGoodsList(ilt):
    tplt = "{:4}\t{:8}\t{:16}"
    print(tplt.format("序列号", "价格", "商品名称"))
    count = 0
    for j in ilt:
        count = count + 1
        print(tplt.format(count, j[0], j[1]))

def main():
    goods = "python爬虫"
    depth = 3
    start_url = 'https://s.taobao.com/search?q=' + goods
    infoList = []
    for i in range(depth):
        try:
            url = start_url + '&s=' + str(44*i)
            html = getHTMLText(url)
            parasePage(infoList, html)
        except:
            continue
    printGoodsList(infoList)

if __name__ == '__main__':
    main()
```

程序运行结果如图 8.31 所示。



序列号	价格	商品名称
1	19.80	Python视频教程Django编程运维开发项目实战
2	19.80	Python视频教程全套编程运维开发项目实战爬虫入门精通 在线课程
3	19.80	python视频教程 零基础2017爬虫django编程运维开发实战在线课程
4	29.00	Python3 Python视频教程自动化运维 爬虫 大数据分析 全栈工程师
5	18.00	Python数据分析与挖掘实战视频教程 网络爬虫 数据处理 分析算法
6	9.99	python3视频Django爬虫项目实战零基础入门到精通2017全新教程
7	9.90	python编程语言零基础入门学习爬虫开发系统培训视频教程
8	100.00	C语言python爬虫留学生.net编程C++算法辅导代写 软件项目 程序咨
9	5.00	Python视频教程分析项目核心数据架构爬虫零基础入门实战视频教程
10	10.00	Python视频教程 初级入门爬虫人工智能深度学习web框架python编程
11	99.00	python人工智能机器学习视频自然语言处理图像处理爬虫视频教程
12	9.90	2017年最新Python3.6网络爬虫实战案例精品高清视频教程
13	50.00	Python网络爬虫工程师视频教程scrapy分布式爬虫豆瓣爬虫微信爬虫
14	15.80	2017全套零基础入门自学Python3视频教程django开发爬虫项目实战
15	18.00	Python爬虫视频教程scrapy分布式爬虫数据分析入门实战视频教程
16	9.90	2018最新零基础python视频教程传智 爬虫 机器学习 人工智能多套
17	5.00	最新python3全栈开发视频教程人工智能大数据分析django爬虫项目
18	15.80	Python3网络分布式爬虫框架Scrapy视频教程数据分析项目实战2018

图 8.31 爬取到有关“Python 爬虫”的商品信息

8.6 Python Web 服务简介



1. Web 服务工作原理

在网络中，安装并运行服务器端程序的计算机称为服务器，而运行客户机端程序的计算机称为客户机。网络通信时，要先有服务器端程序启动，等待客户机端的程序运行并向服务器端口发起连接。一般地，服务器端的程序在一个端口上监听，直到有客户机端的程序发来了连接请求，服务器端随之响应，从而建立起一条数据通信信道。连接过程如图 8.32 所示。

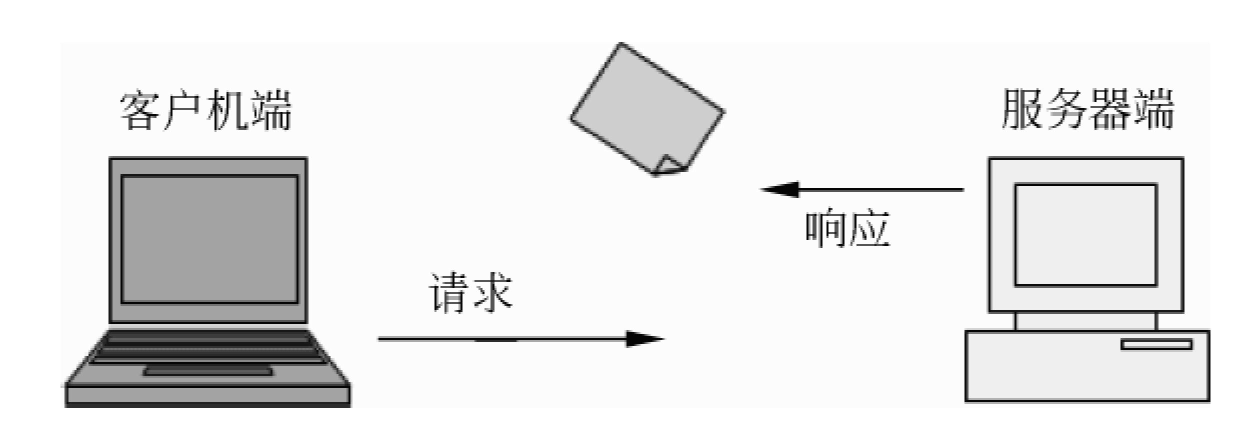


图 8.32　客户机端与服务器端的连接过程

在 Web 服务通信时，其工作可以分解为 4 个过程：

（1）浏览器（客户端）向 Web 服务器端发送一个 HTTP 请求。

（2）Web 服务器收到请求，生成一个 HTML 文档。

（3）Web 服务器把 HTML 文档作为 HTTP 响应发给浏览器。

（4）浏览器收到 HTTP 响应，将页面显示到屏幕上。

2. WSGI 接口

WSGI（Web Server Gateway Interface）是一个 Web 服务网关接口，是将 Python 服务器端程序连接到 Web 服务器的通用协议，其作用如图 8.33 所示。

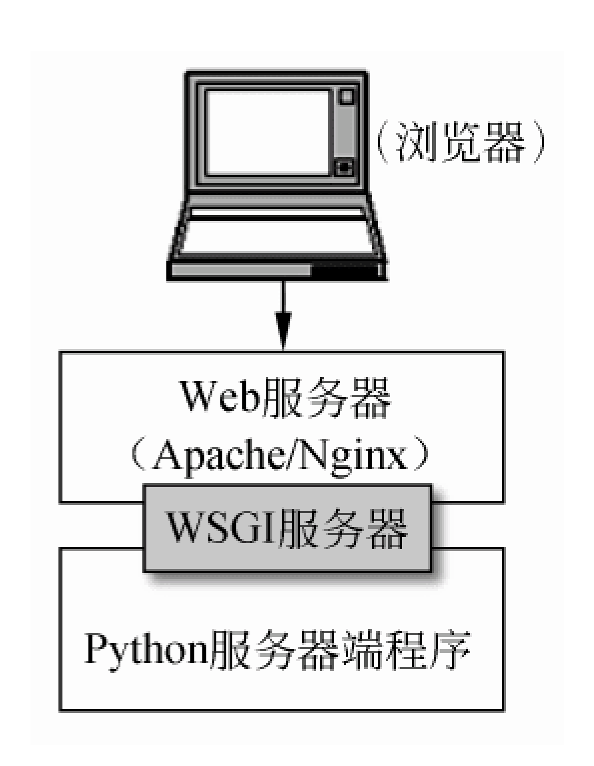


图 8.33　WSGI 连接服务器端程序与 Web 服务器

从图 8.33 中可以看到，WSGI 的接口有两个：一个是与 Web 服务器的接口；另一个是与服务器端程序的接口。WSGI 与 Web 服务器的接口是系统定义的，开发者无须关注。而 WSGI 与服务器端程序的接口是 Python Web 开发者需要学习和掌握的。

3. 编写 WSGI 的接口

例如，下面是一个 WSGI 的接口的简单示例，说明编写 WSGI 的接口的一般方法。

程序代码如下：

```
def show_web(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    return '<h1>Hello, Python web!</h1>'
```

在该 WSGI 的接口示例中，show_web()函数是符合 WSGI 标准的一个 HTTP 处理函数。其参数说明如下：

- environ：包含 HTTP 请求信息的 dict 字典对象。
- start_response：发送 HTTP 响应的函数。

show_web()函数说明：

start_response('200 OK', [('Content-Type', 'text/html')])

发送 HTTP 响应的 Header，由于 Header 只能发送一次，这就意味着 start_response 函数只能执行一次。

该函数的参数：'200 OK'是 HTTP 响应码参数，[('Content-Type', 'text/html')]表示 HTTP Header。

start_response 是一个回调函数，WSGI 的接口程序通过调用 start_response，将 response headers/status 返回给 WSGI 服务器。

函数的返回值 return '<h1>Hello,web!</h1>'作为 HTTP 响应文档发送给服务器。

至于接收 HTTP 请求、解析 HTTP 请求、发送 HTTP 请求等操作都交由 WSGI 服务器完成，WSGI 接口只负责业务逻辑。

4. Python WSGI 服务器端程序

Python 内置了一个 WSGI 服务器，这个模块叫作 wsgiref。因此，在编写 WSGI 服务器端程序时，需要引用 wsgiref 模块。

在 wsgiref 模块中定义了一个 make_server()函数，该函数用于创建 WSGI 服务器。

wsgiref 模块的 make_server()函数创建的 WSGI 服务器没有考虑运行效率，只能用于开发和测试使用。

5. Python Web 网络框架

为了能够快速开发 Python Web 应用项目，目前都是采用 Python 网络框架的开发方式。Python 主流的网络框架有 4 种：Django、Tornado、Flask 和 Twisted。关于 Python Web 框架的深入探讨，超出了本书的范围，请读者自行参考相关专题书籍。

6. Python Web 服务程序设计实例

【例 8-17】 Python Web 服务示例。

（1）编写 WSGI 接口函数的程序 webapp.py

程序代码如下：

```
# webapp.py
def show_web(environ, start_response):
    start_response('200 OK', [('Content-Type', 'text/html')])
    data = ' <meta charset="utf-8"> ' \
          + ' <h1>Hello,  Python  web!  </h1> '\
```

```
        + ' <h1>Python是最流行的计算机编程语言。</h1> '

    return [data.encode('utf-8')]
```

（2）编写 Python 服务器端程序 server.py

程序代码如下：

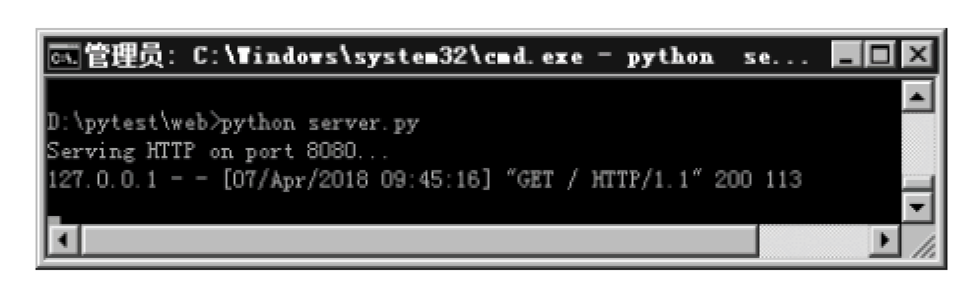
```
# server.py
# 从wsgiref模块导入
from wsgiref.simple_server import make_server
# 从webapp模块中导入编写的show_web()函数
from webapp import show_web

# 创建一个服务器，IP地址为空，端口8000，处理函数是show_web()
httpd = make_server('', 8080, show_web)
print('Serving HTTP on port 8080…')
# 开始监听HTTP请求：
httpd.serve_forever()
```

在命令窗口输入运行服务器端程序的命令：

```
python server.py
```

则启动 WSGI 服务器。再打开浏览器，输入 http://localhost:8080/，可以看到运行结果，如图 8.34 所示。



（a）输入运行程序命令 python server.py



（b）在浏览器输入 http://localhost:8080

图 8.34　Web 服务运行结果

习　题　8

1．编写程序，实现端口数据转发功能。
2．编写程序，实现端口重定向功能。
3．参考例 8-3，编写一个 FTP 客户端程序，实现上传、下载、删除、更名等功能。
4．参考例 8-13，爬取当地天气预报信息。
5．参考例 8-14，爬取购物网站中“华为手机”商品信息。

第 9 章　算法设计及机器学习实战入门

9.1　常见的数据结构



视频录像

9.1.1　堆栈

堆栈是一个后进先出的数据结构，其工作方式就像生活中常见的爬直梯子，先爬上梯子的人肯定最后下梯子。

下面设计一个类，用列表来存放栈中的元素信息，利用列表的 append()和 pop()方法可以实现栈元素的入栈 push 和出栈 pop 的操作，list.append(obj)是向列表添加一个对象 obj，list.pop(index=−1)是删除指定位置的对象，默认删除最后一个对象，也就是说，list.pop()是删除列表中下标最大的元素。

【例 9-1】 设计一个模拟堆栈结构的程序。

程序代码如下：

```
# 堆栈，后进先出
class Stack():
    def __init__(self,size):
        self.size=size
        self.stack=[]
        self.top=-1

    def push(self,x):          # 入栈之前检查栈是否已满
        if self.isfull():
            raise exception("stack is full")
        else:
            self.stack.append(x)
            self.top=self.top+1

    def pop(self):             # 出栈之前检查栈是否为空
        if self.isempty():
            raise exception("stack is empty")
        else:
            self.top=self.top-1
            self.stack.pop()
```

```
    def isfull(self):
        return self.top+1 == self.size
    def isempty(self):
        return self.top == '-1'
    def showStack(self):
        print(self.stack)

s=Stack(10)
for i in range(8):
    s.push(i)
print('入栈元素顺序： 先入 <--- 后入')
s.showStack()

for i in range(3):
    s.pop()
print('\n 出栈元素顺序：  先入 --> 后入')
s.showStack()
```

在本例的 Stack 类中设有一个 top 属性，用来指示栈的存储情况，初始值为 1，一旦插入一个元素，其值加 1，利用 top 的值判定栈是空还是满。

执行程序时，先将 0，1，2，3，4，5，6，7 依次入栈，然后删除栈顶的前三个元素。程序运行结果如下：

```
入栈元素顺序： 先入 <--- 后入
[0, 1, 2, 3, 4, 5, 6, 7]

出栈元素顺序：  先入 --> 后入
[0, 1, 2, 3, 4]
```

9.1.2 队列

1. 队列结构

队列是一种先进先出的数据类型，它的原理类似于顾客在超市收银处排队，排在队列前面的人先于排在后面的人接受服务。

新的元素通过入队的方式添加到队列的末尾，而出队就是将队列的头元素删除。

下面设计一个类，用列表存放栈中元素的信息，利用列表的 append()和 pop()方法实现队列的入队 enqueue 和出队 dequeue 的操作。

【例 9-2】 设计一个模拟队列结构的程序。

程序代码如下：

```
# 队列，先进先出
class Queue():

    def __init__(self,size):
```

```
        self.size = size
        self.front = -1
        self.rear = -1
        self.queue = []

    def enqueue(self,ele):         # 入队操作
        if self.isfull():
            raise exception("queue is full")
        else:
            self.queue.append(ele)
            self.rear = self.rear + 1

    def dequeue(self):             # 出队操作
        if self.isempty():
            raise exception("queue is empty")
        else:
            self.queue.pop(0)
            self.front = self.front+1

    def isfull(self):
        return self.rear-self.front+1 == self.size
    def isempty(self):
        return self.front == self.rear
    def showQueue(self):
        print(self.queue)

q = Queue(10)
for i in range(9):
    q.enqueue(i)
print('进入队列元素顺序： 先入 <--- 后入')
q.showQueue()
for i in range(3):
    q.dequeue()
print('\n 出队列元素顺序： 先入 ---> 后入')
q.showQueue()
print(q.isempty())
```

在本例的 Queue 类中设置了两个属性 front 和 rear，用于模拟队列的头尾指针，通过这两个指针值之间的关系，可以判定队列是空还是满。

程序运行结果如下：

```
进入队列元素顺序： 先入 <--- 后入

[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
 出队列元素顺序： 先入 <--- 后入
[3, 4, 5, 6, 7, 8]
False
```

2. 队列 queue 模块

在 Python 系统中定义了 queue 模块，使得队列的应用更加高效。

（1）创建 “队列” 对象

应用 queue 模块的 Queue 类可以创建队列对象：

```
import queue
q = queue.Queue(maxsize = 10)
```

queue.Queue 类定义的队列长度可由构造函数的可选参数 maxsize 设定。如果 maxsize 小于 1 就表示队列长度无限。

（2）将数据加入到队列中

Queue 类的 put(item)方法把数据加入到队列的末尾。

例如：

```
q.put(150)
```

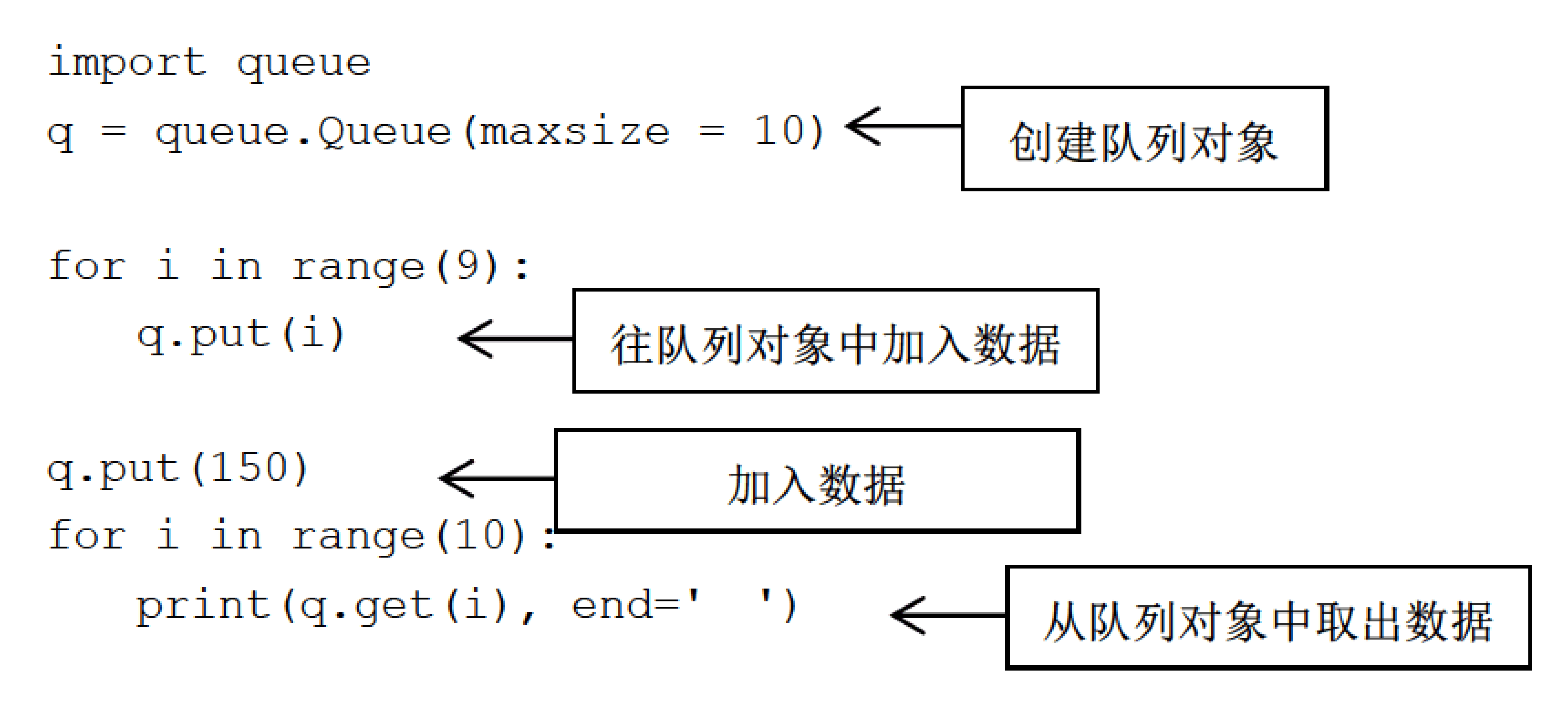
将数据 150 加入到队列的最后。

（3）从队列中取出数据

Queue 类的 get()方法把排在队列最前面（队头）的数据从队列中删除，并返回该数据。

【例 9-3】 队列模块使用示例。

程序代码如下：


```
import queue
q = queue.Queue(maxsize = 10)

for i in range(9):
    q.put(i)

q.put(150)
for i in range(10):
    print(q.get(i), end='  ')
```


9.1.3 链表

链表由一系列在内存中不一定连续存放的数据对象构成，这些数据对象按线性顺序排序。每个对象含有表数据和指向后继数据对象的指针。最后一个对象的指针指向 Null。为了方便链表的删除与插入操作，通常在链表的最前面添加一个链表头。链表的结构如图 9.1 所示。

对链表进行操作时，如删除一个节点，只需要将前一节点项的指针跳过该节点项，指向后一节点项，如图 9.2 所示。

图 9.1　链表结构示意图

若要在链表中插入一个节点项，在操作上要执行两次指针的调整，让前一节点指针指向新节点，而新节点的指针指向后一节点，如图 9.3 所示。

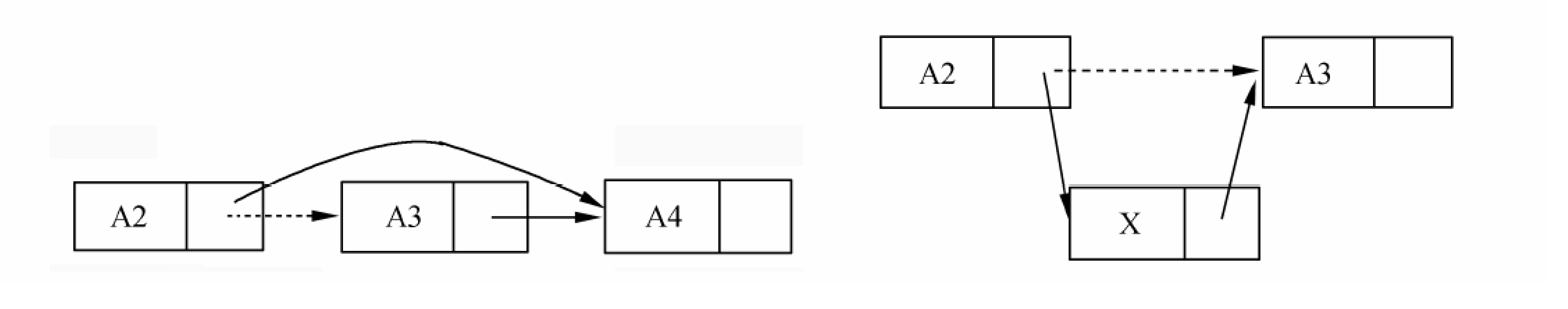


图 9.2　删除节点　　　　图 9.3　插入节点

在链表中，每个节点 Node 可以分为两部分：一部分为数据；另一部分为指针，指向下一个节点 Node。

下面定义一个链表的节点类 Node。

其中：

_data：存放节点的数据。

_next：节点指针，存放下一个节点对象。

程序代码如下：

```
class Node():
  __slots__ = ['_data','_next']
  def __init__(self,data):
    self._data = data
    self._next = None

  def getdata(self):              # 获取节点数据
    return self._data

  def getNext(self):              # 获取指向的下一节点
    return self._next

  def setdata(self,newdata):      # 设置节点数据
    self._data = newdata

  def setNext(self,newnext):      # 设置指向新节点
    self._next = newnext
```

1. 定义链表类 SingleLinkedList 及构建函数

链表类定义了构造函数。在构造函数中定义了链表头_head 属性。

链表类 SingleLinkedList 及构造函数的定义如下：

```
class SingleLinkedList():
  def __init__(self):
    self._head = None      #初始化为空链表
```

2. 检测链表是否为空

程序代码如下：

```
def isEmpty(self):
    return self._head == None
```

3. 在链表前端添加元素

程序代码如下：

```
def add(self,data):
  temp = Node(data)
  temp.setNext(self._head)
  self._head = temp
```

4. 在链表尾部添加元素

程序代码如下：

```
def append(self,data):
  temp = Node(data)
  if self.isEmpty():
    self._head = temp  #若为空表，将添加的元素设为第一个元素
  else:
    current = self._head
    while current.getNext()!=None:
      current = current.getNext()  #遍历链表
    current.setNext(temp)  #此时current为链表最后的元素
```

5. 检索某数据是否在链表中

程序代码如下：

```
def search(self,data):
  current = self._head
  founddata = False
  while current != None and not founddata:
    if current.getdata() == data:
      founddata = True
    else:
      current = current.getNext()
  return founddata
```

6. 索引某数据在链表中的位置

程序代码如下：

```
def index(self,data):
  current = self._head
  count = 0
  found = None
  while current != None and not found:
    count += 1
    if current.getdata() == data:
      found = True
    else:
      current = current.getNext()
  if found:
    return count
  else:
    raise '%s is not in linkedlist'%data
```

7. 删除链表中的某项数据

程序代码如下：

```
def remove(self,data):
  current = self._head
  pre = None
  while current != None:
    if current.getdata() == data:
      if not pre:
        self._head = current.getNext()
      else:
        pre.setNext(current.getNext())
      break
    else:
      pre = current
      current = current.getNext()
```

8. 在链表中插入数据

程序代码如下：

```
def insert(self,pos,item):
    if posself.size():
        self.append(item)
    else:
        temp = Node(item)
        count = 1
        pre = None
        current = self._head
```

```
    while count
```

【例 9-4】 编写一个基本功能完整的链表程序。

程序代码如下：

```
# 定义节点类
class Node():
  __slots__ = ['_data','_next']
  def __init__(self,data):
    self._data = data
    self._next = None

  def getdata(self):
    return self._data

  def getNext(self):
    return self._next

  def setdata(self,newdata):
    self._data = newdata

  def setNext(self,newnext):
    self._next = newnext

# 定义链表类
class SingleLinkedList():
  def __init__(self):
    self._head = None      # 初始化为空链表

  def isEmpty(self):
    return self._head == None

  def size(self):
    current = self._head
    count = 0
    while current != None:
      count += 1
      current = current.getNext()
    return count

  def travel(self):
    current = self._head
    while current != None:
      print( current.getdata())
      current = current.getNext()
```

```
def add(self,data):
  temp = Node(data)
  temp.setNext(self._head)
  self._head = temp

def append(self,data):
  temp = Node(data)
  if self.isEmpty():
    self._head = temp                # 若为空表，将添加的数据设为第一个节点
  else:
    current = self._head
    while current.getNext() != None:
      current = current.getNext()    # 遍历链表
    current.setNext(temp)            # 此时current为链表最后的节点

def search(self,data):
  current = self._head
  founddata = False
  while current != None and not founddata:
    if current.getdata() == data:
      founddata = True
    else:
      current = current.getNext()
  return founddata

def index(self,data):
  current = self._head
  count = 0
  found = None
  while current != None and not found:
    count += 1
    if current.getdata() == data:
      found = True
    else:
      current  =current.getNext()
  if found:
    return count
  else:
    raise '%s is not in linkedlist'%data

def remove(self,data):
  current = self._head
  pre = None
  while current != None:
    if current.getdata() == data:
```

```
            if not pre:
              self._head = current.getNext()
            else:
              pre.setNext(current.getNext())
            break
          else:
            pre = current
            current = current.getNext()

    def insert(self,pos,data):
      if pos <= 1:
        self.add(data)
      elif pos>self.size():
        self.append(data)
      else:
        temp = Node(data)
        count = 1
        pre = None
        current = self._head
        while count < pos:
          count += 1
          pre = current
          current = current.getNext()
        pre.setNext(temp)
        temp.setNext(current)

  if __name__ == '__main__':
    a = SingleLinkedList()
    for i in range(1,10):
      a.append(i)
    print(a.size())
    a.travel()
    print(a.search(6))
    print(a.index(5))
    a.remove(4)
    a.travel()
    a.insert(4,100)
    a.travel()
```

9.1.4 树

1. 树结构

树结构在计算机科学应用广泛，操作系统、图形学、数据库、计算机网络等领域均要使用树的结构和处理方法。数据结构中的树和真正的树有许多相似之处，也包括根、树枝和叶子，它们的不同在于计算机中树的根在顶层而它的叶子在底部。简单的树结构如图 9.4

所示。

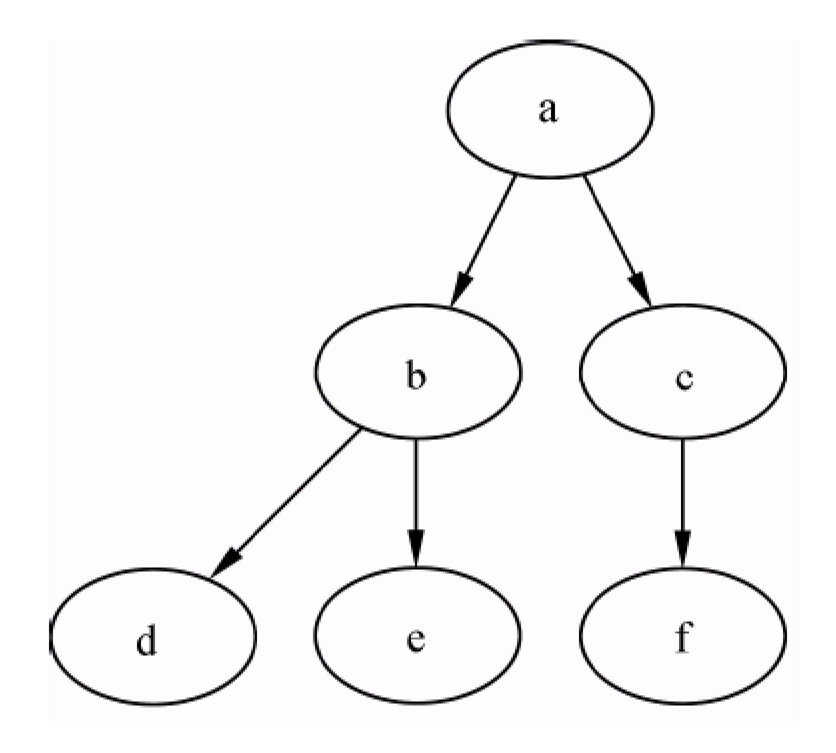


图 9.4　简单的树结构

树有以下基本概念。

（1）节点（Node）

图 9.4 中的 a、b、c、d、e、f 均是树的节点。

（2）根节点（Root）

图 9.4 中的 a 为树的根节点。

（3）叶子节点（LeafNode）

图 9.4 中的 d、e、f 为树的叶子节点。

（4）内部节点（InternalNode）

图 9.4 中的 b、c 为树的内部节点。

2. 建立树模型的方法

建立树模型的第一步，是要将表达式字符串分解成符号保存在列表里。

这里有 4 种符号需要考虑：左括号、右括号、操作符和操作数。当读到一个左括号时，将开始一个新的表达式，这时要创建一个子树来对应这个新的表达式；相反，当读到一个右括号，就得结束这个表达式。另外，操作数是叶子节点，操作数所属的操作符是子节点，且每个操作符都应该有一个左子节点和一个右子节点。

（1）规则

将上述分析总结如下建立树模型的规则：

① 如果当前读入的符号是左括号‘(’，则添加一个新的节点作为当前节点的左子节点，并下降到左子节点处。

② 如果当前读入的字符在列表[‘+’，‘-’，‘/’，‘*’]中，则将读入的字符设置为当前节点的值。添加一个新的节点作为当前节点的右子节点，并下降到右子节点处。

③ 如果当前读入的字符是一个数字，则将该数字设置为当前节点的值，并返回到它的父节点。

④ 如果当前读入的字符是右括号‘)’，则返回当前节点的父节点。

例如，设有表达式 (3+(4*5))，现将其分解为如下的字符列表：

```
[ '(', '3', '+', '(', '4', '*', '5' ,')',')' ]
```

（2）建立树模型

下面根据该字符列表建立树模型。

① 创建一个仅包括一个空根节点的树，如图 9.5 所示。

② 读入的第一个字符左括号‘(’，根据规则 ①，创建一个新的节点作为当前节点的左子节点，并将当前节点变为这个新的子节点，如图 9.6 所示。

③ 读入数字 3，根据规则③，将当前节点的值赋值为 3，然后返回当前节点的父节点，当前节点成为叶子节点，如图 9.7 所示。

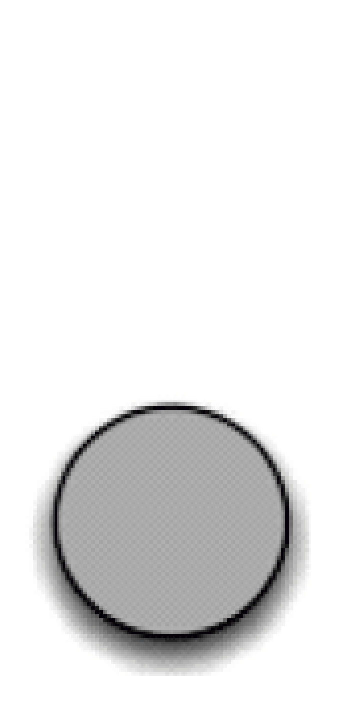


图 9.5　空的根节点

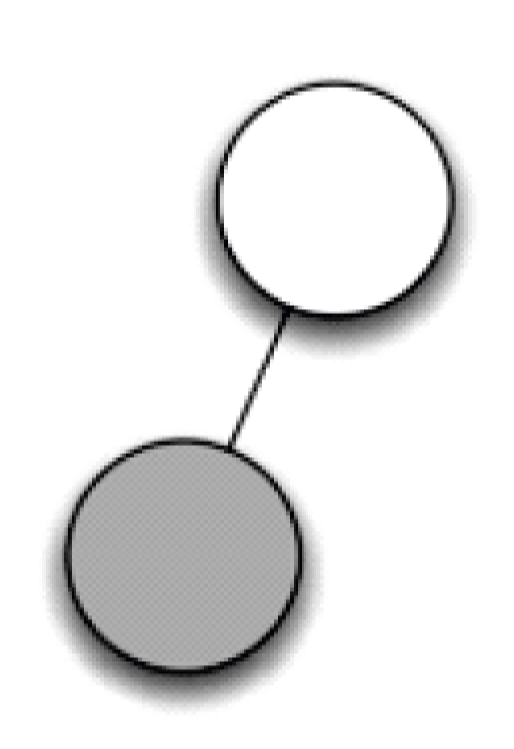


图 9.6 “(”号创建左子节点

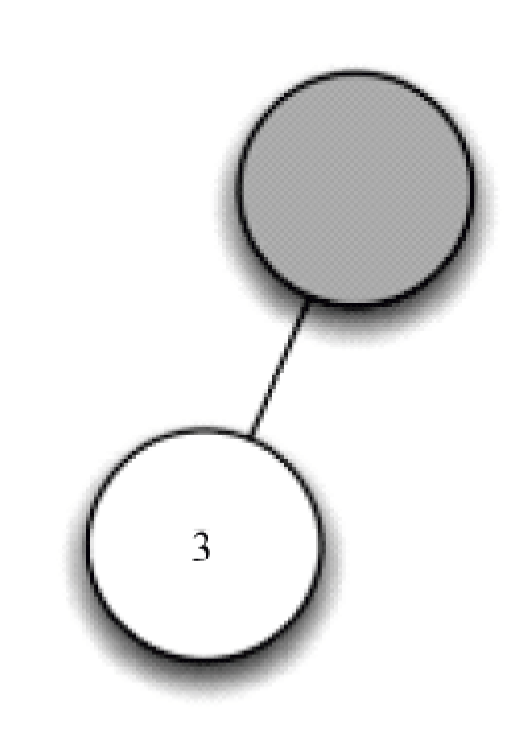


图 9.7　数字 3 为叶子节点

④ 读入字符加号‘+’。根据规则 ②，将当前节点（根节点）的值赋值为+，然后添加一个新的节点作为其右子节点，并且将当前节点变为这个新的子节点，如图 9.8 所示。

⑤ 读入字符左括号‘(’。根据规则 ①，创建一个新的节点作为当前节点的左子节点，并将当前节点变为这个新的子节点，如图 9.9 所示。

⑥ 读入数字 4。根据规则 ③，将当前节点的值赋值为 4，然后返回当前节点的父节点，当前节点成为叶子节点，如图 9.10 所示。

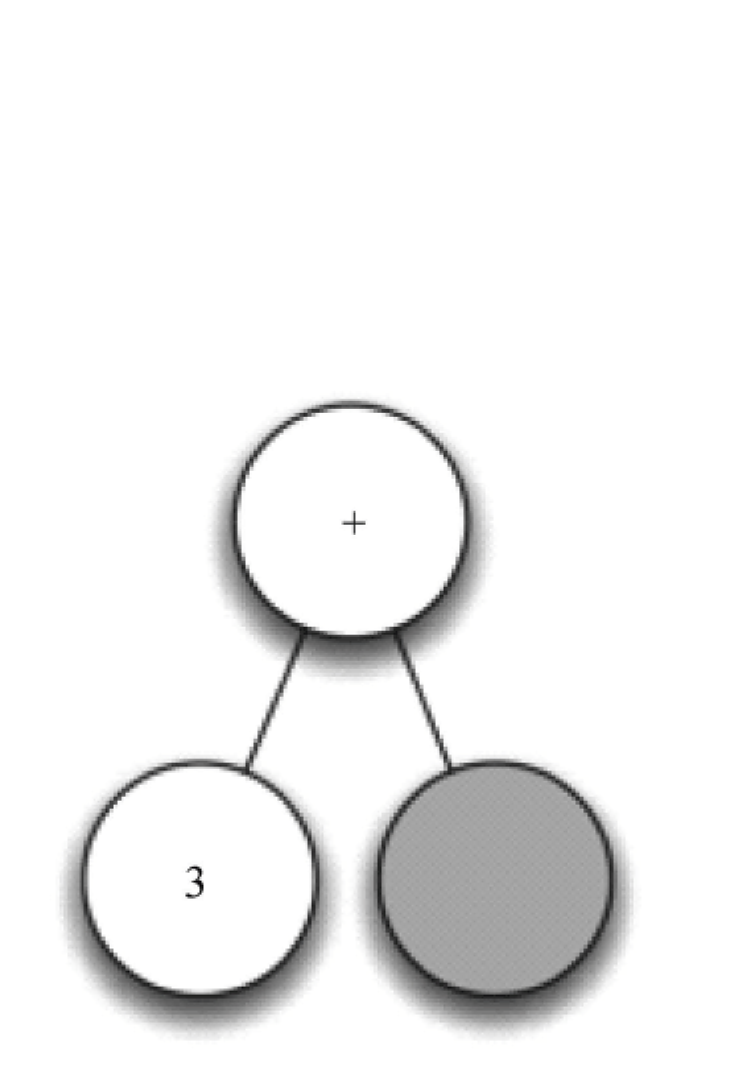


图 9.8 ‘+’号则创建右子节点

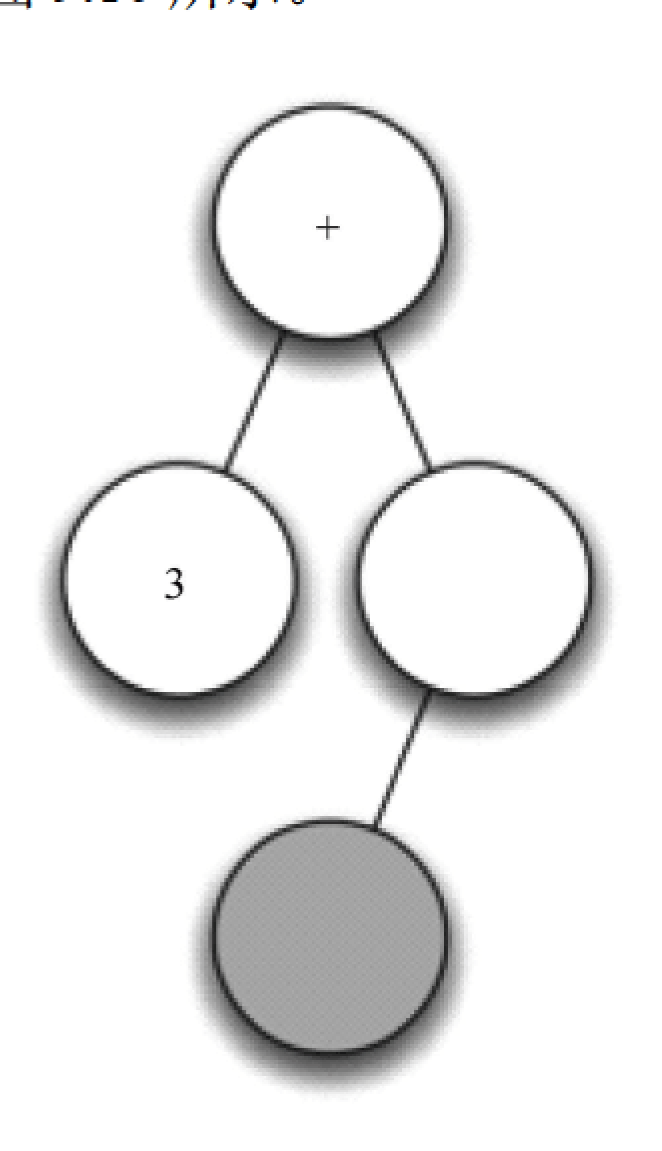


图 9.9 ‘(’ 创建左子节点

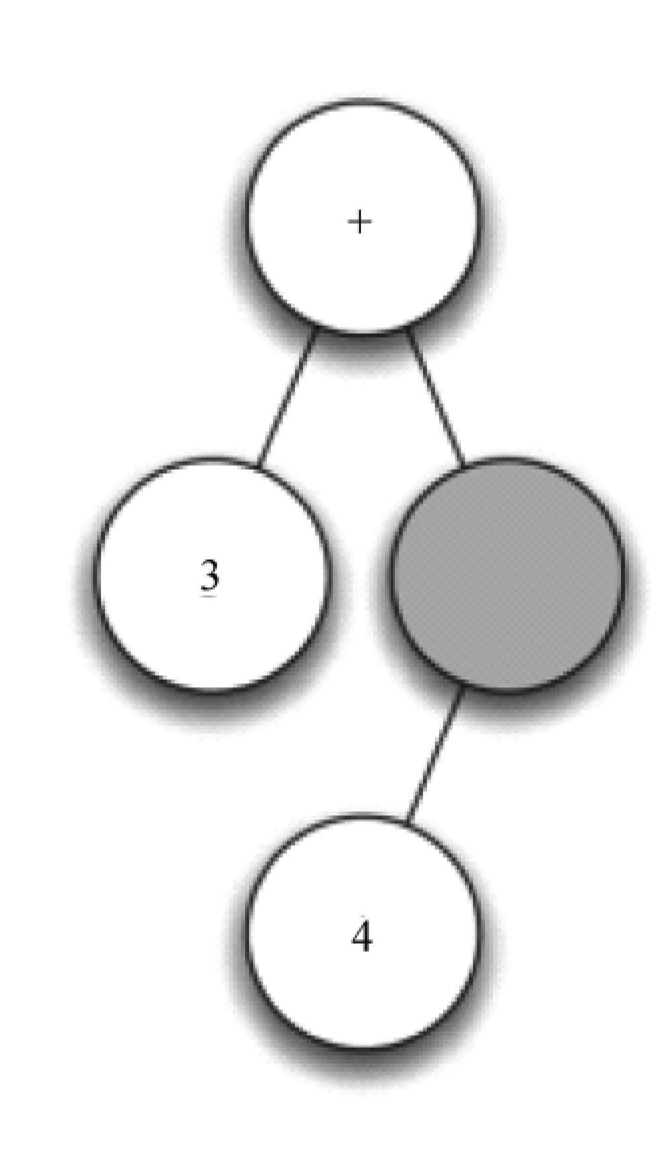


图 9.10　数字 4 为叶子节点

⑦ 读入字符乘号‘*’。根据规则 ②，将当前节点的值赋值为‘*’，然后添加一个新的节点作为其右子节点，并且将当前节点变为这个新的子节点，如图 9.11 所示。

⑧ 读入数字 5，根据规则③，将当前节点的值赋值为 5，然后返回当前节点的父节点，如图 9.12 所示。

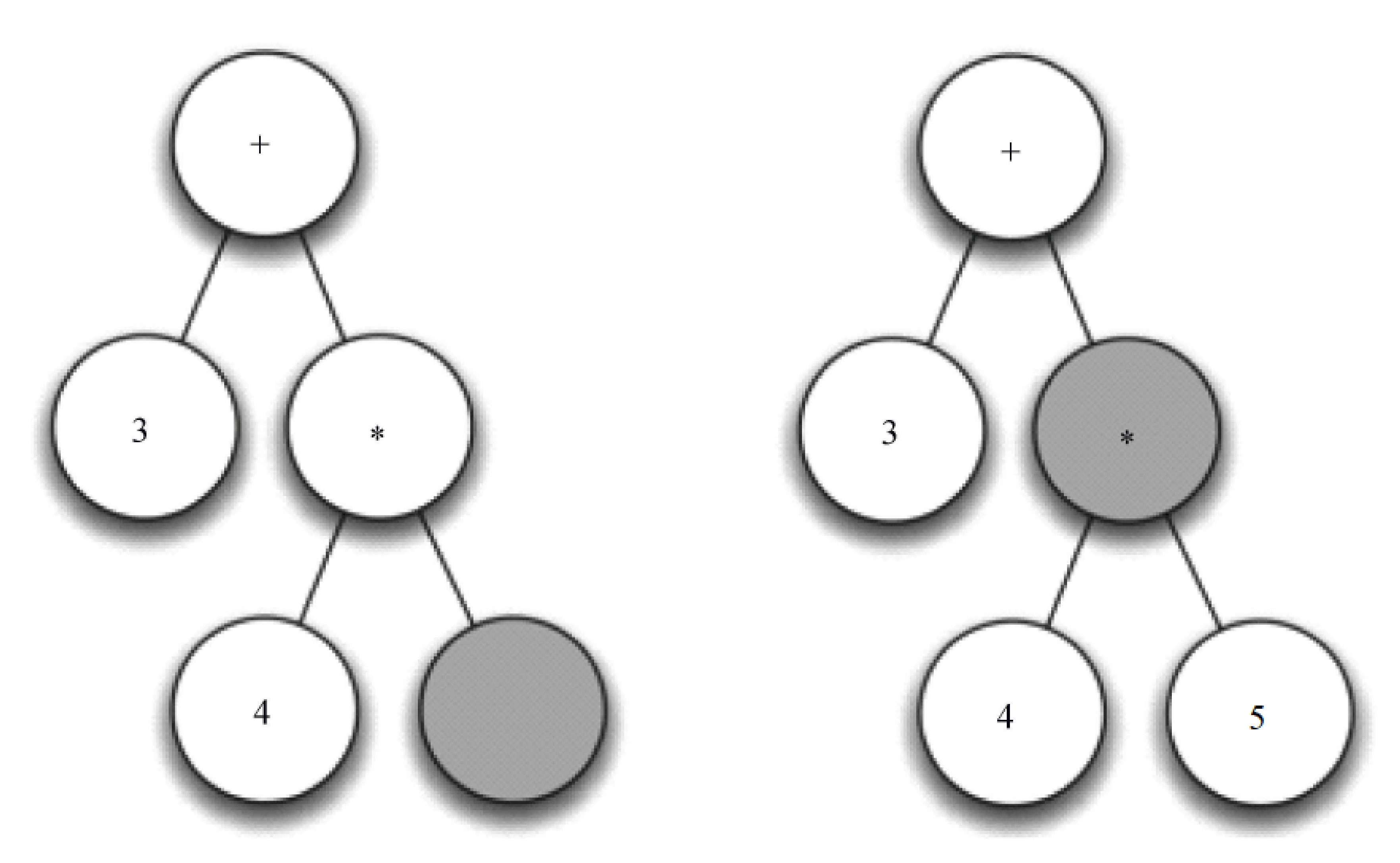


图 9.11 ‘*’号则创建右子节点　　　图 9.12 数字 5 为叶子节点

⑨ 读入字符右括号 ‘)’。根据规则④，返回到当前节点的父节点，即返回到*节点。

⑩ 再次读入一个右括号 “)”。根据规则④，返回到当前节点的父节点，即返回到+节点（根节点）。

3. 编写建立树模型的程序

上面通过 10 个步骤，创建了树的模型。下面根据刚才的步骤编写生成树结构模型的程序代码。

（1）安装 pythonds 模块

编写树的程序，需要用到 pythonds 模块，可以使用 pip 来安装 pythonds 模块。其命令格式如下：

```
pip  install  pythonds
```

（2）编写程序，建立一个树的模型

根据建立树模型的规则，编写建立树的程序。

【例 9-5】 编程，建立表达式((10 + 5) * 3)的树模型。

程序代码如下：

```
from pythonds.basic.stack import Stack
from pythonds.trees.binaryTree import BinaryTree

def buildParseTree(fpexp):
    fplist = fpexp.split()
    pStack = Stack()
    eTree = BinaryTree('')
    pStack.push(eTree)
```

```
    currentTree = eTree
    for i in fplist:
        if i == '(':
            currentTree.insertLeft('')
            pStack.push(currentTree)
            currentTree = currentTree.getLeftChild()
        elif i not in ['+', '-', '*', '/', ')']:
            currentTree.setRootVal(int(i))
            parent = pStack.pop()
            currentTree = parent
        elif i in ['+', '-', '*', '/']:
            currentTree.setRootVal(i)
            currentTree.insertRight('')
            pStack.push(currentTree)
            currentTree = currentTree.getRightChild()
        elif i == ')':
            currentTree = pStack.pop()
        else:
            raise ValueError
    return eTree

pt = buildParseTree("( ( 10 + 5 ) * 3 )")
pt.postorder()
```


程序运行结果如下：

```
10
5
+
3
*
```

4. 树的遍历

树的遍历分为先序遍历、中序遍历、后序遍历。

- 先序遍历：先访问根节点，然后再访问左子节点，最后访问右子节点。
- 中序遍历：先访问左子节点，然后再访问根节点，最后访问右子节点。
- 后序遍历：先访问左子节点，然后再访问右子节点，最后访问根节点。

设树的节点为 TreeNode，则节点类可以写成：

```
class TreeNode(object):
    def __init__(self):
        self.data = '#'
        self.left_child = None    # 左子节点
        self.right_child = None   # 右子节点
```

【例 9-6】 树的遍历示例。

程序代码如下：

```
class TreeNode(object):
    def __init__(self):
        self.data = '#'
        self.left_child = None
        self.right_child = None
```

定义节点类

```
class Tree(TreeNode):
    # create a tree
    def create_tree(self, tree):
        data = input('->')
        if data == '#':
            tree = None
        else:
            tree.data = data
            tree.left_child = TreeNode()
            self.create_tree(tree.left_child)
            tree.right_child = TreeNode()
            self.create_tree(tree.right_child)
```

从键盘输入数据

‘#’ 代表空

建立各个节点，构建树

```
    # visit为一个节点
    def visit(self, tree):
        # 输入#号代表空树
        if tree.data is not '#':
            print( str(tree.data) + '\t',)

    # 先序遍历
    def pre_order(self, tree):
        if tree is not None:
            self.visit(tree)
            self.pre_order(tree.left_child)
            self.pre_order(tree.right_child)

    # 中序遍历
    def in_order(self, tree):
        if tree is not None:
            self.in_order(tree.left_child)
            self.visit(tree)
            self.in_order(tree.right_child)

    # 后序遍历
    def post_order(self, tree):
        if tree is not None:
            self.post_order(tree.left_child)
            self.post_order(tree.right_child)
            self.visit(tree)
```

```
t = TreeNode()
tree = Tree()
tree.create_tree(t)
print('先序遍历：')
tree.pre_order(t)
print( '\n')
print('中序遍历：')
tree.in_order(t)
print( '\n')
print('后序遍历：')
tree.post_order(t)
```

运行程序，从键盘输入数据 5、4、#、#、6、#、# 。程序运行结果如下：

```
->5
->4
->#
->#
->6
->#
->#
先序遍历：
5  4  6
中序遍历：
4  5  6
后序遍历：
4  6  5
```


视频录像

9.2　迷宫问题算法设计

用一个二维数组表示一个简单的迷宫，用 0 表示通路，用 1 表示阻断，老鼠在每个点上可以移动相邻的东南西北 4 个点，设计一个算法，模拟老鼠走迷宫，找到从入口到出口的一条路径，如图 9.13 所示。

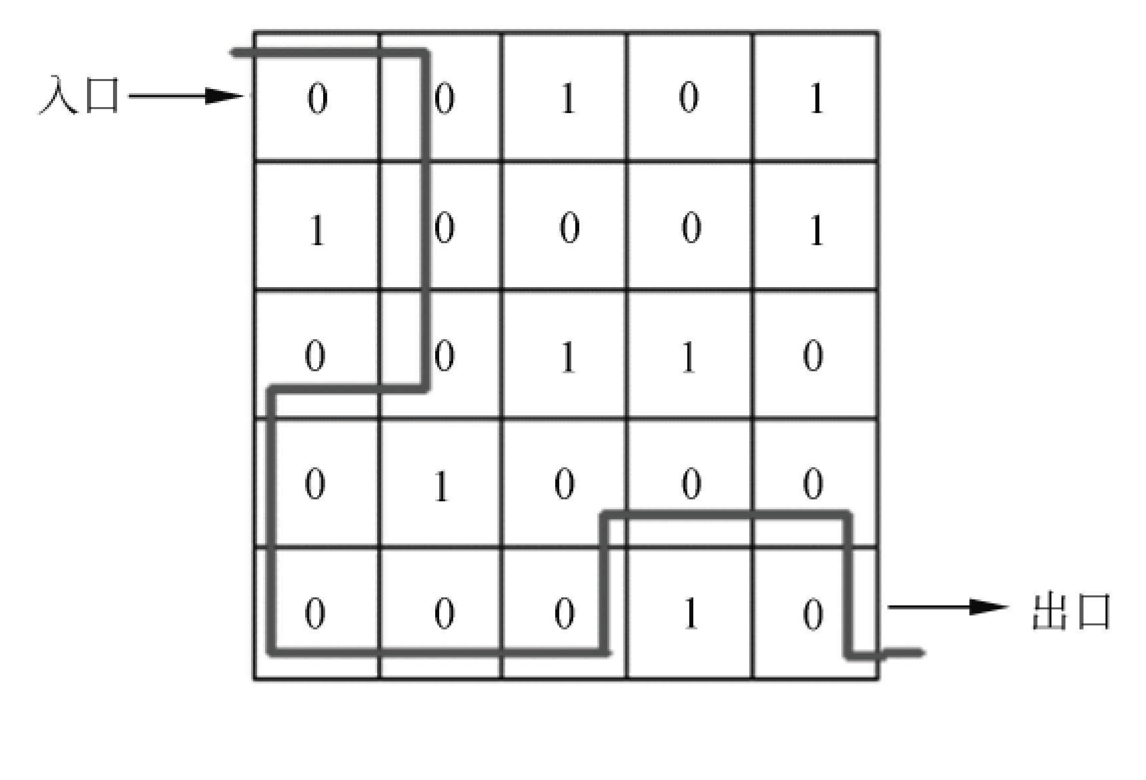


图 9.13　迷宫问题

【例 9-7】 迷宫问题解法 1：回溯算法

设计思路：

① 首先用一个列表 source 存储迷宫图，一个列表 route_stack 存储路线图，一个列表 route_history 存储走过的点，起点（0,0），终点（4,4）。

② 老鼠在每个点都有上下左右 4 种方案可选，需要定义这些方案的执行方法。

③ 最后做一个循环，如果当前点不是（4,4）的话就依次执行上下左右 4 种方法，但是有些限制，如尝试走过的点不会再尝试走，（0，x）点无法再执行向上的方法等。

程序代码如下：

```
route_stack = [[0,0]]
route_history = [[0,0]]
source=[[0,0,1,0,1],[1,0,0,0,1],[0,0,1,1,0],[0,1,0,0,0],[0,0,0,1,0]]
def up(location):
    # 横坐标为0，无法再向上走
    if location[1] == 0:
        return False
    else:
        new_location = [location[0],location[1]-1]
        # 已经尝试过的点不会尝试第二次
        if new_location in route_history:
            return False
        # 碰到墙不走
        elif source[new_location[0]][new_location[1]] == 1:
            return False
        else:
            route_stack.append(new_location)
            route_history.append(new_location)
            return True

def down(location):
    if location[1] == 4:
        return False
    else:
        new_location = [location[0],location[1]+1]
        if new_location in route_history:
            return False
        elif source[new_location[0]][new_location[1]] == 1:
            return False
        else:
            route_stack.append(new_location)
            route_history.append(new_location)
            return True

def left(location):
```

```
    if location[0] == 0:
        return False
    else:
        new_location = [location[0]-1,location[1]]
        if new_location in route_history:
            return False
        elif source[new_location[0]][new_location[1]] == 1:
            return False
        else:
            route_stack.append(new_location)
            route_history.append(new_location)
            return True

def right(location):
    if location[0] == 4:
        return False
    else:
        new_location = [location[0]+1,location[1]]
        if new_location in route_history:
            return False
        elif source[new_location[0]][new_location[1]] == 1:
            return False
        else:
            route_stack.append(new_location)
            route_history.append(new_location)
            return True
lo = [0,0]
while route_stack[-1] != [4,4]:
    if up(lo):
        lo = route_stack[-1]
        continue
    if down(lo):
        lo = route_stack[-1]
        continue
    if left(lo):
        lo = route_stack[-1]
        continue
    if right(lo):
        lo = route_stack[-1]
        continue
    route_stack.pop()
    lo = route_stack[-1]
print(route_stack)
```

程序运行结果如下：

```
[[0, 0], [0, 1], [1, 1], [2, 1], [2, 0], [3, 0], [4, 0], [4, 1], [4, 2],
[3, 2], [3, 3], [3, 4], [4, 4]]
```

【例 9-8】 迷宫问题解法 2：递归实现走迷宫。

在一个由 0 或 1 构成的二维数组中，假设 0 是可以移动到的点，1 是不能移动到的点，如何从数组中间一个值为 0 的点出发，每次只能朝上下左右 4 个方向移动一个单位，当移动到二维数组的边缘，即可得到问题的解，类似的问题都可以称为迷宫问题。

设计思路：

对于数组中的一个点，该点的 4 个方向可以通过横纵坐标的加减轻松表示，当移动到一个可移动点时，整个问题又变为和初始状态一样的问题，继续搜索 4 个方向找可以移动的点，直到移动到数组的边缘。

程序代码如下：

```
# 判断坐标的有效性，如果超出数组边界或是不满足值为0的条件，说明该点无效返回# False，
# 否则返回True
import sys
def valid(maze,x,y):
    if (x>=0 and x<len(maze) and y>=0 and y<len(maze[0]) and maze[x][y]==0):
        return True
    else:
        return False
# 移步函数实现
def walk(maze,x,y):
    # 如果位置是迷宫的出口，说明成功走出迷宫
    if(x==0 and y==0):
        print("successful!")
        sys.exit (1)
    # 递归主体实现
    if valid(maze,x,y):
        # print(x,y)
        maze[x][y]=2          # 做标记，防止折回
        # 针对4个方向依次试探，如果失败，撤销一步
        if not walk(maze,x,y-1):
            maze[x][y]=0      # 回溯到原位置
        elif not walk(maze,x-1,y):
            maze[x][y]=0
        elif not walk(maze,x+1,y):
            maze[x][y]=0
        elif not walk(maze,x,y+1):
            maze[x][y]=0
        else:
            Print("无路可走")
            return False      # 无路可走说明，没有解
    return True
maze=[[0, 0,1,0,1],
      [1,0,0,0,1],
      [0,0,1,1,0],
      [0,1,0,0,0],
      [0,0,0,1,0]]
```

```
[[0, 0], [0, 1], [1, 1], [2, 1], [2, 0], [3, 0], [4, 0], [4, 1], [4, 2],
[3, 2], [3, 3], [3, 4], [4, 4]]
```

【例 9-8】 迷宫问题解法 2：递归实现走迷宫。

在一个由 0 或 1 构成的二维数组中，假设 0 是可以移动到的点，1 是不能移动到的点，如何从数组中间一个值为 0 的点出发，每次只能朝上下左右 4 个方向移动一个单位，当移动到二维数组的边缘，即可得到问题的解，类似的问题都可以称为迷宫问题。

设计思路：

对于数组中的一个点，该点的 4 个方向可以通过横纵坐标的加减轻松表示，当移动到一个可移动点时，整个问题又变为和初始状态一样的问题，继续搜索 4 个方向找可以移动的点，直到移动到数组的边缘。

程序代码如下：

```
# 判断坐标的有效性，如果超出数组边界或是不满足值为0的条件，说明该点无效返回# False，
# 否则返回True
import sys
def valid(maze,x,y):
    if (x>=0 and x<len(maze) and y>=0 and y<len(maze[0]) and maze[x][y]==0):
        return True
    else:
        return False
# 移步函数实现
def walk(maze,x,y):
    # 如果位置是迷宫的出口，说明成功走出迷宫
    if(x==0 and y==0):
        print("successful!")
        sys.exit (1)
    # 递归主体实现
    if valid(maze,x,y):
        # print(x,y)
        maze[x][y]=2            # 做标记，防止折回
        # 针对4个方向依次试探，如果失败，撤销一步
        if not walk(maze,x,y-1):
            maze[x][y]=0        # 回溯到原位置
        elif not walk(maze,x-1,y):
            maze[x][y]=0
        elif not walk(maze,x+1,y):
            maze[x][y]=0
        elif not walk(maze,x,y+1):
            maze[x][y]=0
        else:
            Print("无路可走")
            return False        # 无路可走说明，没有解
    return True
maze=[[0, 0,1,0,1],
      [1,0,0,0,1],
      [0,0,1,1,0],
      [0,1,0,0,0],
      [0,0,0,1,0]]
```

```
walk(maze,4,4)           # 从（4, 4）出发
```

程序运行结果如下：

```
4 4
3 4
3 3
3 2
4 2
4 1
4 0
3 0
2 0
2 1
1 1
0 1
0 0
successful!
```



视频录像

9.3 曲线点抽稀算法

在处理矢量化数据时，记录中往往会有很多重复数据，对进一步数据处理带来诸多不便。多余的数据一方面浪费了较多的存储空间；另一方面造成要表达的图形不光滑或不符合标准。因此要通过某种规则，在保证矢量曲线形状不变的情况下， 最大限度地减少数据点个数，这个过程称为抽稀。

曲线点抽稀算法就是对曲线进行采样简化。即在曲线上取有限个点，将其变为折线，并能够在一定程度保持原有形状。图 9.14 和图 9.15 所示为地图坐标数据抽稀前和抽稀后的结果对比。

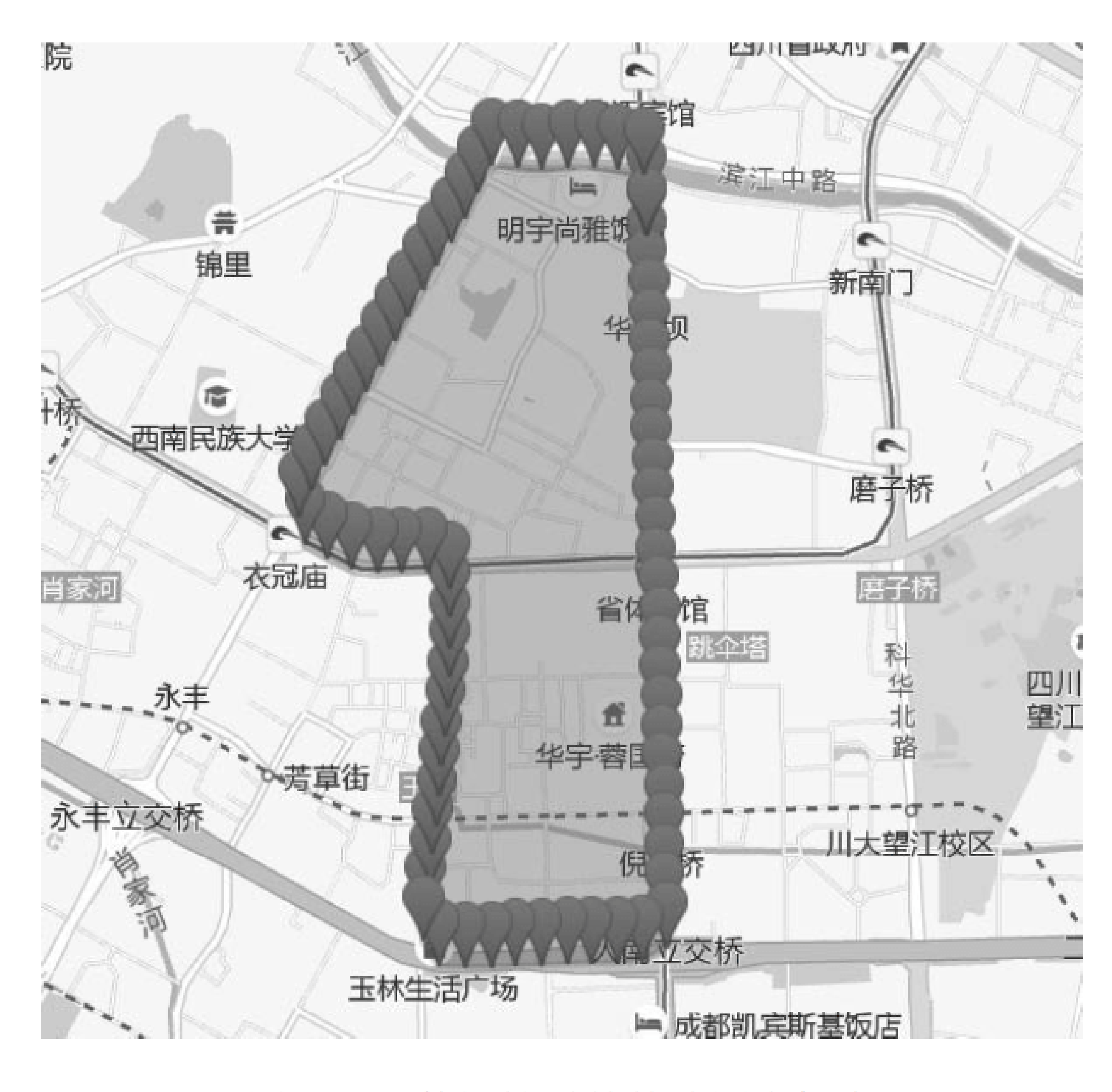


图 9.14 数据抽稀前的地图坐标点

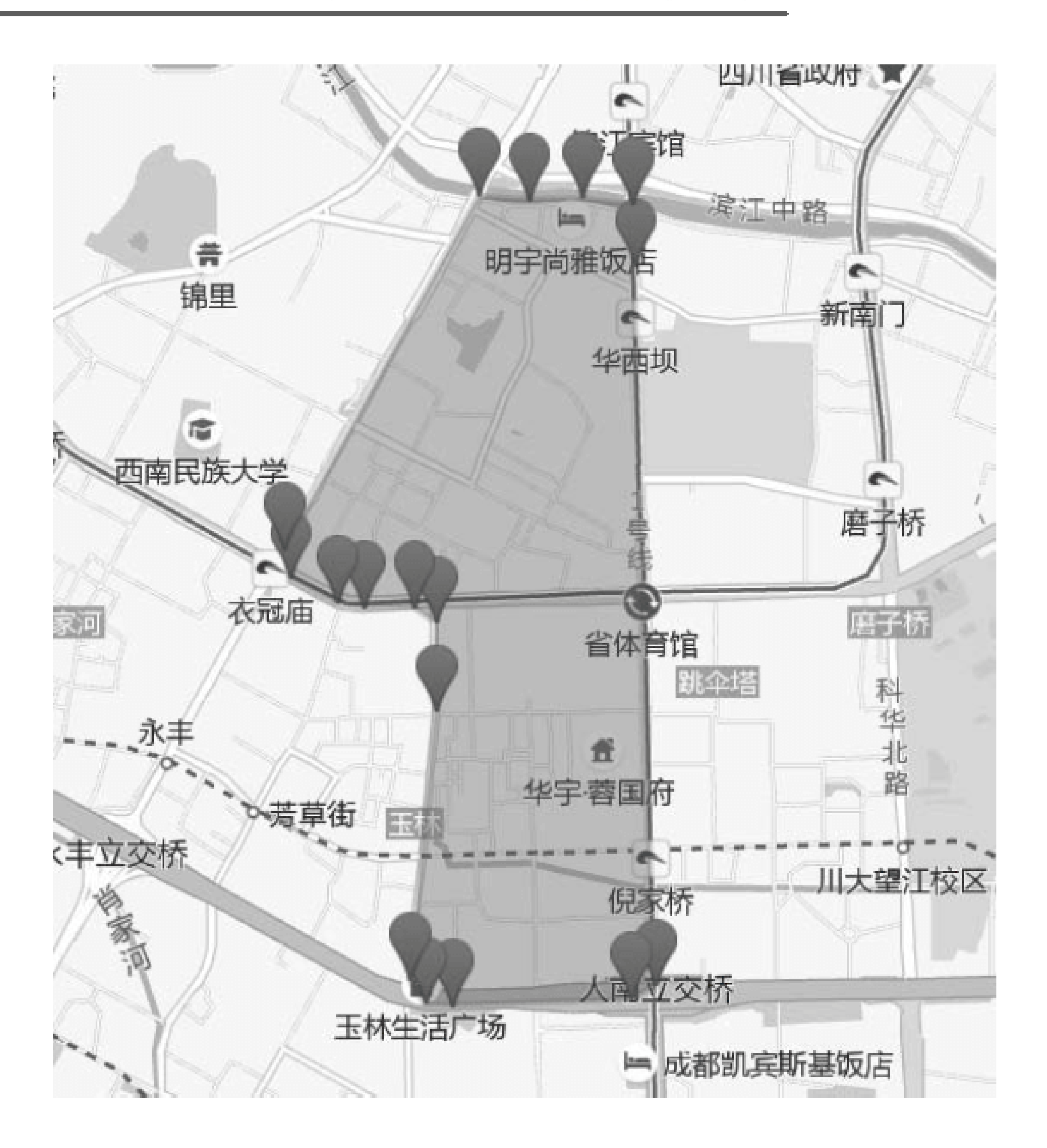


图 9.15　数据抽稀后的地图坐标点

比较常用的曲线点抽稀算法有两种：道格拉斯-普克（Douglas-Peuker）算法和垂距限值法。

9.3.1　道格拉斯-普克算法

道格拉斯-普克（Douglas-Peuker，DP）算法的算法过程如下：

① 连接曲线首尾两点 A、B。

② 依次计算曲线上所有点到 A、B 两点所在曲线的距离。

③ 计算最大距离 D，如果 D 小于阈值 threshold，则去掉曲线上出 A、B 外的所有点；如果 D 大于阈值 threshold，则把曲线以最大距离分割成两段。

④ 对所有曲线分段重复步骤①～③，直到所有 D 均小于阈值，即完成抽稀。

DP 算法的抽稀精度与阈值有很大关系，阈值越大，简化程度越大，点减少得越多；反之，简化程度越低，点保留得越多，形状也越趋于原曲线。

【例 9-9】 实现曲线点抽稀的道格拉斯-普克算法。

程序代码如下：

```
from __future__ import division
from math import sqrt, pow

THRESHOLD = 0.0001  # 阈值


def point2LineDistance(point_a, point_b, point_c):
    """
    :param point_a:
    :param point_b:
    :param point_c:
```

```
        :return:
        """
        # 计算直线的斜率和截距
        if point_b[0] == point_c[0]:
            return 9999999
        slope = (point_b[1] - point_c[1]) / (point_b[0] - point_c[0])
        intercept = point_b[1] - slope * point_b[0]

        # 计算点到直线的距离
        distance=abs(slope*point_a[0]-point_a[1]+intercept)/sqrt(1+pow(slope,2))
        return distance

    class DouglasPeuker(object):
        def __init__(self):
            self.threshold = THRESHOLD
            self.qualify_list = list()
            self.disqualify_list = list()

        def diluting(self, point_list):
            """
            抽稀
            :param point_list:二维点列表
            :return:
            """
            if len(point_list) < 3:
                self.qualify_list.extend(point_list[::-1])
            else:
                # 找到与收尾两点连线距离最大的点
                max_distance_index, max_distance = 0, 0
                for index, point in enumerate(point_list):
                    if index in [0, len(point_list) - 1]:
                        continue
                    distance = point2LineDistance(point, point_list[0],
                                point_list[-1])
                    if distance > max_distance:
                        max_distance_index = index
                        max_distance = distance

                # 若最大距离小于阈值，则去掉所有中间点
                # 反之，则将曲线按最大距离点分割
                if max_distance < self.threshold:
                    self.qualify_list.append(point_list[-1])
                    self.qualify_list.append(point_list[0])
                else:
                    # 将曲线按最大距离的点分割成两段
```

```
                sequence_a = point_list[:max_distance_index]
                sequence_b = point_list[max_distance_index:]

                for sequence in [sequence_a, sequence_b]:
                    if len(sequence) < 3 and sequence == sequence_b:
                        self.qualify_list.extend(sequence[::-1])
                    else:
                        self.disqualify_list.append(sequence)

    def main(self, point_list):
        self.diluting(point_list)
        while len(self.disqualify_list) > 0:
            self.diluting(self.disqualify_list.pop())
        print(self.qualify_list)
        print('抽稀后数据个数：',len(self.qualify_list) )

if __name__ == '__main__':
  d = DouglasPeuker()
  d.main([
    [104.066228, 30.644527], [104.066279, 30.643528], [104.066296, 30.642528],
    [104.066314, 30.641529], [104.066332, 30.640529], [104.066383, 30.639530],
    [104.066400, 30.638530], [104.066451, 30.637531], [104.066468, 30.636532],
    [104.066518, 30.635533], [104.066535, 30.634533], [104.066586, 30.633534],
    [104.066636, 30.632536], [104.066686, 30.631537], [104.066735, 30.630538],
    [104.066785, 30.629539], [104.066802, 30.628539], [104.066820, 30.627540],
    [104.066871, 30.626541], [104.066888, 30.625541], [104.066906, 30.624541],
    [104.066924, 30.623541], [104.066942, 30.622542], [104.066960, 30.621542],
    [104.067011, 30.620543], [104.066122, 30.620086], [104.065124, 30.620021],
    [104.064124, 30.620022], [104.063124, 30.619990], [104.062125, 30.619958],
    [104.061125, 30.619926], [104.060126, 30.619894], [104.059126, 30.619895],
    [104.058127, 30.619928], [104.057518, 30.620722], [104.057625, 30.621716],
    [104.057735, 30.622710], [104.057878, 30.623700], [104.057984, 30.624694],
    [104.058094, 30.625688], [104.058204, 30.626682], [104.058315, 30.627676],
    [104.058425, 30.628670], [104.058502, 30.629667], [104.058518, 30.630667],
    [104.058503, 30.631667], [104.058521, 30.632666], [104.057664, 30.633182],
    [104.056664, 30.633174], [104.055664, 30.633166], [104.054672, 30.633289],
    [104.053758, 30.633694], [104.052852, 30.634118], [104.052623, 30.635091],
    [104.053145, 30.635945], [104.053675, 30.636793], [104.054200, 30.637643],
    [104.054756, 30.638475], [104.055295, 30.639317], [104.055843, 30.640153],
    [104.056387, 30.640993], [104.056933, 30.641830], [104.057478, 30.642669],
    [104.058023, 30.643507], [104.058595, 30.644327], [104.059152, 30.645158],
    [104.059663, 30.646018], [104.060171, 30.646879], [104.061170, 30.646855],
    [104.062168, 30.646781], [104.063167, 30.646823], [104.064167, 30.646814],
    [104.065163, 30.646725], [104.066157, 30.646618], [104.066231, 30.645620],
    [104.066247, 30.644621],   ])
```

运行程序结果如下：

```
[[104.066247, 30.644621], [104.066157, 30.646618], [104.065163, 30.646725],
[104.060171, 30.646879], [104.059663, 30.646018], [104.052623, 30.635091],
[104.052852, 30.634118], [104.054672, 30.633289], [104.055664, 30.633166],
[104.057664, 30.633182], [104.058521, 30.632666], [104.058503, 30.631667],
[104.058425, 30.62867], [104.058315, 30.627676], [104.057518, 30.620722],
[104.058127, 30.619928], [104.059126, 30.619895], [104.066122, 30.620086],
[104.067011, 30.620543], [104.06696, 30.621542], [104.066228, 30.644527]]
抽稀后数据个数：21
```

9.3.2 垂距限值算法

垂距限值法其实和 DP 算法原理一样，但是垂距限值不是从整体角度考虑，而是依次扫描每一个点，检查是否符合要求。

垂距限值法的算法过程如下：

① 以第二个点开始，计算第二个点到前一个点和后一个点所在直线的距离 d。

② 如果 d 大于阈值，则保留第二个点，计算第三点到第二点和第四点所在直线的距离 d；若 d 小于阈值则舍弃第二点，计算第三点到第一点和第四点所在直线的距离 d。

③ 依次类推，直至曲线上倒数第二点。

【例 9-10】 编写实现垂距限值算法的程序。

程序代码如下：

```
from __future__ import division
from math import sqrt, pow

THRESHOLD = 0.0001  # 阈值

def point2LineDistance(point_a, point_b, point_c):
    """
    :param point_a:
    :param point_b:
    :param point_c:
    :return:
    """
    # 计算直线的斜率和截距
    if point_b[0] == point_c[0]:
        return 9999999
    slope = (point_b[1] - point_c[1]) / (point_b[0] - point_c[0])
    intercept = point_b[1] - slope * point_b[0]

    # 计算点到直线的距离
    distance = abs(slope*point_a[0]-point_a[1]+intercept)/sqrt(1+pow(slope,2))
    return distance
```

```
class LimitVerticalDistance(object):
    def __init__(self):
        self.threshold = THRESHOLD
        self.qualify_list = list()

    def diluting(self, point_list):
        """
        抽稀
        :param point_list:二维点列表
        :return:
        """
        self.qualify_list.append(point_list[0])
        check_index = 1
        while check_index < len(point_list) - 1:
            distance = point2LineDistance(point_list[check_index],
                                          self.qualify_list[-1],
                                          point_list[check_index + 1])
            if distance < self.threshold:
                check_index += 1
            else:
                self.qualify_list.append(point_list[check_index])
                check_index += 1
        return self.qualify_list

if __name__ == '__main__':
    limit = LimitVerticalDistance()
    diluting = limit.diluting([[104.066228, 30.644527], [104.066279, 30.643528],
    [104.066296, 30.642528], [104.066314, 30.641529], [104.066332, 30.640529],
    [104.066383, 30.639530], [104.066400, 30.638530], [104.066451, 30.637531],
    [104.066468, 30.636532], [104.066518, 30.635533], [104.066535, 30.634533],
    [104.066586, 30.633534], [104.066636, 30.632536], [104.066686, 30.631537],
    [104.066735, 30.630538], [104.066785, 30.629539], [104.066802, 30.628539],
    [104.066820, 30.627540], [104.066871, 30.626541], [104.066888, 30.625541],
    [104.066906, 30.624541], [104.066924, 30.623541], [104.066942, 30.622542],
    [104.066960, 30.621542], [104.067011, 30.620543], [104.066122, 30.620086],
    [104.065124, 30.620021], [104.064124, 30.620022], [104.063124, 30.619990],
    [104.062125, 30.619958], [104.061125, 30.619926], [104.060126, 30.619894],
    [104.059126, 30.619895], [104.058127, 30.619928], [104.057518, 30.620722],
    [104.057625, 30.621716], [104.057735, 30.622710], [104.057878, 30.623700],
    [104.057984, 30.624694], [104.058094, 30.625688], [104.058204, 30.626682],
    [104.058315, 30.627676], [104.058425, 30.628670], [104.058502, 30.629667],
    [104.058518, 30.630667], [104.058503, 30.631667], [104.058521, 30.632666],
    [104.057664, 30.633182], [104.056664, 30.633174], [104.055664, 30.633166],
    [104.054672, 30.633289], [104.053758, 30.633694], [104.052852, 30.634118],
    [104.052623, 30.635091], [104.053145, 30.635945], [104.053675, 30.636793],
```

```
[104.054200, 30.637643], [104.054756, 30.638475], [104.055295, 30.639317],
[104.055843, 30.640153], [104.056387, 30.640993], [104.056933, 30.641830],
[104.057478, 30.642669], [104.058023, 30.643507], [104.058595, 30.644327],
[104.059152, 30.645158], [104.059663, 30.646018], [104.060171, 30.646879],
[104.061170, 30.646855], [104.062168, 30.646781], [104.063167, 30.646823],
[104.064167, 30.646814], [104.065163, 30.646725], [104.066157, 30.646618],
[104.066231, 30.645620], [104.066247, 30.644621], ])
print('抽稀后数据个数：',len(diluting))
print(diluting)
```

程序运行结果如下：

```
抽稀后数据个数： 13
[[104.066228, 30.644527], [104.067011, 30.620543], [104.066122, 30.620086],
[104.058127, 30.619928], [104.057518, 30.620722], [104.058518, 30.630667],
[104.058521, 30.632666], [104.057664, 30.633182], [104.054672, 30.633289],
[104.052852, 30.634118], [104.052623, 30.635091], [104.060171, 30.646879],
[104.066157, 30.646618]]
```

比较 DP 算法和垂距限值法，它们的原理其实是一样。DP 算法是从整体上考虑一条完整的曲线，实现时较垂距限值法复杂，但垂距限值法可能会在某些情况下导致局部最优。另外，在实际使用中发现采用点到另外两点所在直线距离的方法判断偏离，在曲线弧度比较大的情况下比较准确。如果在曲线弧度比较小，弯曲程度不明显时，这种方法抽稀效果不是很理想，建议使用三点所围成的三角形面积作为判断标准。

9.4 Python 机器学习实战入门


视频录像

9.4.1 机器学习及其算法

机器学习是一门既“古老”又“新兴”的计算机科学技术，属于人工智能研究与应用的分支。机器学习是一种通过利用数据，训练出模型，然后使用模型预测的一种方法。

1. 机器学习的特点

机器学习有如下特点。

机器学习与人类思考与推理相比较，人类的思考与推理是根据历史经验总结出某种规律进而推导出结果，而机器学习是计算机利用已有的数据，建立数据模型，并利用此模型预测出未知结果。

机器学习的过程与人类对历史经验归纳总结的过程如图 9.16 所示。

① 机器学习系统要解决的问题都是无法直接使用固定规则或流程代码完成的问题，通常这类问题对人类来说很简单。例如，人类可以非常容易从一张照片中区分出人和猫，而机器很难做到。

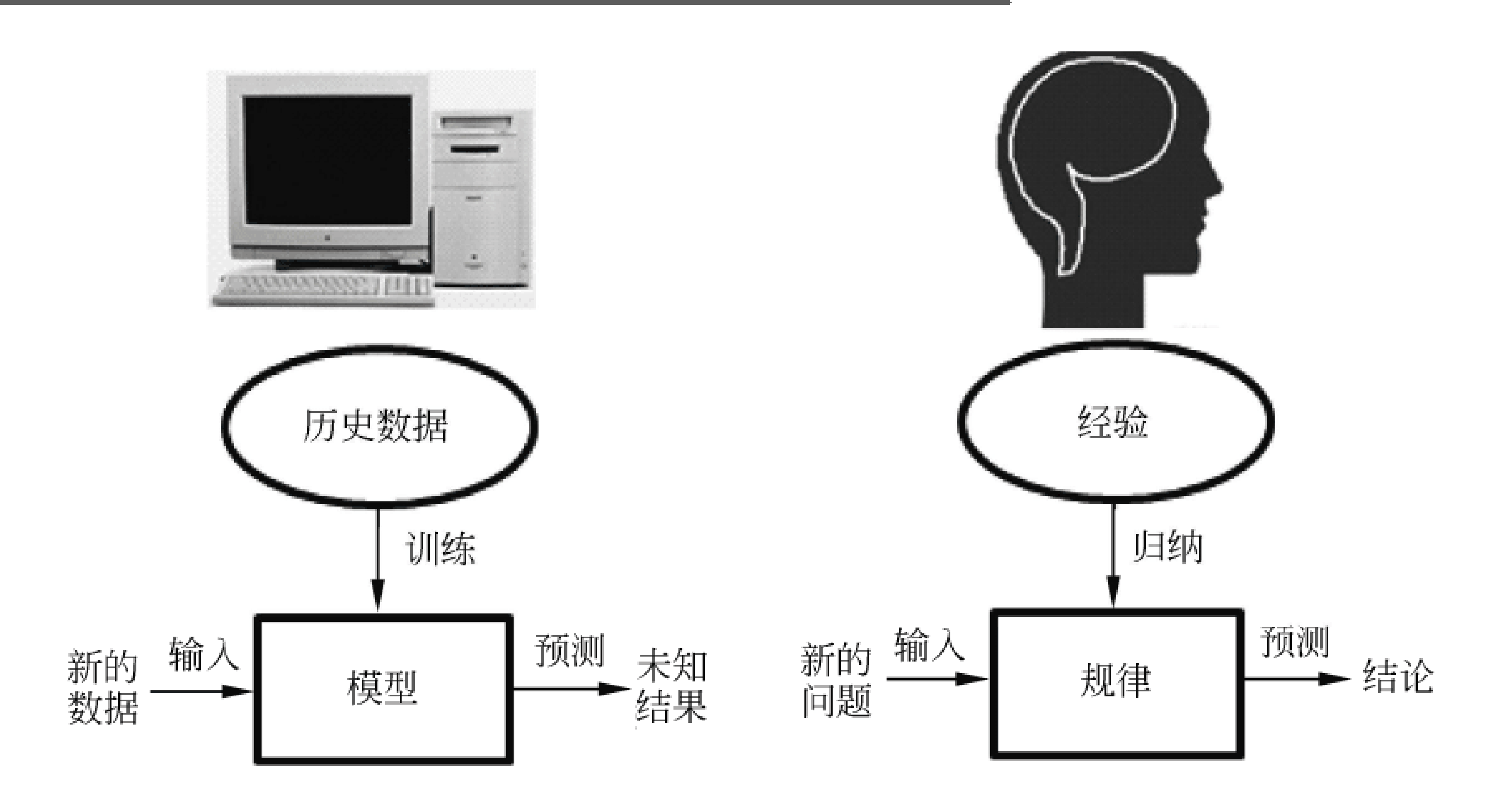


图 9.16　机器学习与人类思考的比较

② 机器学习系统具有学习能力是指它能够不断地从经验和数据中吸取经验教训，从而应对未来的预测任务。机器学习的核心是统计和归纳。

③ 机器学习系统具备不断改善自身对应具体任务的能力。

2. 机器学习的算法分类

机器学习的算法有三种：监督式学习、非监督式学习和强化学习。下面对这三种机器学习的算法进行简要介绍。

（1）监督式学习算法

监督式学习的算法机制为：算法由目标变量或结果变量（或因变量）组成。这些变量由已知的一系列预示变量（自变量）预测而来。利用这一系列变量，生成一个将输入值映射到期望输出值的函数。这个训练过程会一直持续，直到模型在训练数据上获得期望的精确度。监督式学习的例子有回归、决策树、随机森林、K-近邻算法、逻辑回归等。

（2）非监督式学习算法

非监督式学习的算法机制：在这个算法中，没有任何目标变量或结果变量要预测或估计。这个算法用在数据降维和聚类问题分析。这种分析方式被广泛地用来细分客户，根据干预的方式分为不同的用户组。非监督式学习的例子有关联算法和 K-均值算法。

（3）强化学习算法

强化学习的算法机制：机器被放在一个能让它通过反复试错来训练自己的环境中。机器从过去的经验中进行学习，并且尝试利用了解最透彻的知识作出精确的商业判断。应用这个算法可以训练机器进行决策。强化学习的例子有马尔可夫决策过程。

3. 机器学习的常用算法

机器学习的常用算法如下：

- 线性回归；
- 逻辑回归；
- 决策树；
- 神经网络；
- SVM（支持向量机算法）；

- 朴素贝叶斯；
- K 最近邻算法；
- K 均值算法；
- 随机森林算法；
- 降维算法；
- Gradient Boost 和 AdaBoost 算法。

关于机器学习涉及算法的深入探讨，超出了本书的范围，请读者自行参考相关书籍。

9.4.2 机器学习应用实例

1. 机器学习库 sklearn

scikit-learn 简称 sklearn，是 Python 重要的机器学习库。sklearn 支持包括分类、回归、降维和聚类 4 大机器学习算法，还包含了特征提取、数据处理和模型评估三大模块。应用 sklearn 模块，可以极大提高机器学习的效率。

可以使用 pip 来安装 sklearn 模块。其命令如下：

```
pip install sklearn
```

在程序的前面，需要引用 sklearn 模块：

```
import sklearn
```

2. 决策树 sklearn.tree 类

机器学习库 sklearn 中的决策树 tree 可用于分类决策算法，也可用于回归决策算法。分类决策树的类需要使用 DecisionTreeClassifier()方法。

使用时要添加引用语句：

```
from sklearn import tree
```

3. 机器学习算法应用的主要步骤

应用机器学习库 sklearn 进行决策的主要步骤如图 9.17 所示。

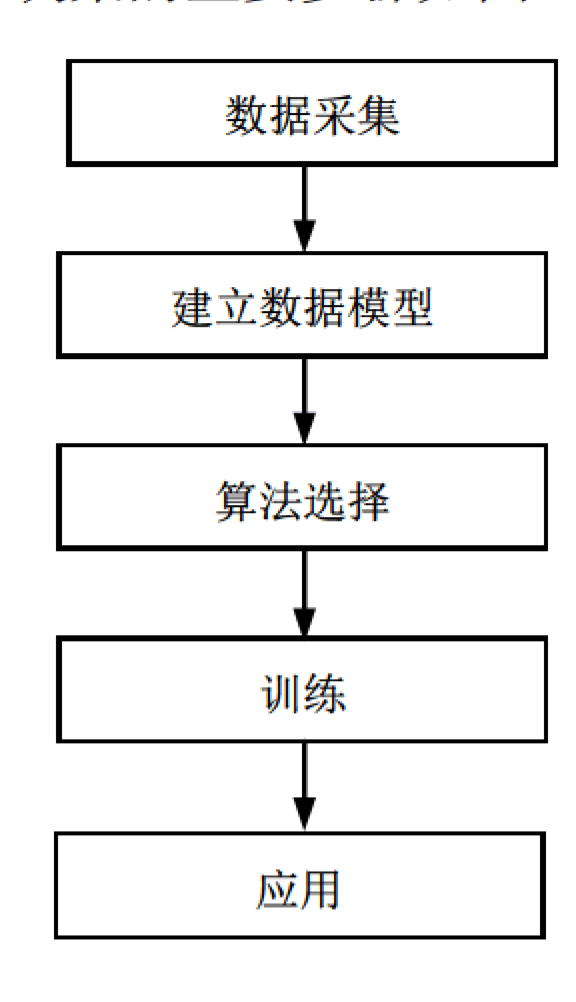


图 9.17 机器学习算法应用的主要步骤

4. 应用实例

【例 9-11】 根据人的特征（身高、胡子）数据，自动判断所属性别。

设有人的性别实验检测数据如表 9.1 所示。

表 9.1 人的性别实验检测数据

性别	身高	胡子（1 代表有，0 代表无）
男	178	1
女	155	0
男	177	0
女	165	0
男	169	1
女	160	0

（1）建立数据模型

根据实验检测数据，得到实验训练数据列表：

```
feature = [ [178,1], [155,0], [177,0], [165, 0], [169,1], [160, 0] ]
```

对应的性别分类列表为：

```
label = ['man', 'woman', 'man', 'woman', 'man', 'woman']
```

（2）创建决策树对象

有了上述数据模型之后，应用机器学习创建决策树对象。

```
clf = tree.DecisionTreeClassifier()
```

（3）自动判断性别分类

决策树对象根据给出一组数据，自动判断属于哪个性别。

```
s1 = clf.predict([ [158, 0] ])          # 对测试点[158, 0]进行预测
print(s1)                               # 输出预测结果值
```

程序代码如下：

```
import sklearn
from sklearn import tree

feature = [[178,1], [155,0], [177,0], [165,0], [169,1], [160,0]]# 训练数据
label = ['man', 'woman', 'man', 'woman', 'man', 'woman']        # 性别分类
clf = tree.DecisionTreeClassifier()     # 分类决策树的分类决策方法
clf = clf.fit(feature, label)           # 拟合训练数据，得到训练模型参数
s1 = clf.predict([ [158, 0] ])          # 对测试点[158, 0]进行预测
s2 = clf.predict([ [176, 1] ])          # 对测试点[176, 1]进行预测
print('s1 = ',  s1)                     # 输出预测结果值
print('s2 = ', s2)                      # 输出预测结果值
```

程序运行结果如下：

```
s1 = ['woman']
s2 = ['man']
```

9.5 机器学习案例精选

视频录像

【例 9-12】 应用机器学习的决策树算法，自动完成信贷审核。

1. 决策树算法简介

决策树算法（decision tree）是一种基本的分类与回归算法。决策树又称为判定树，是运用于分类的一种树结构，其中的每个内部节点代表对某一属性的一次测试，每条边代表一个测试结果，叶子节点代表某个类或类的分布。

决策树的决策过程需要从决策树的根节点开始，待测数据与决策树中的特征节点进行比较，并按照比较结果选择下一比较分支，直到叶子节点作为最终的决策结果。决策树中的每个节点表示对象属性的判断条件，其分支表示符合节点条件的对象。树的叶子节点表示对象所属的预测结果。

2. 经验熵

对信息集合不确定性的度量方式称为信息熵或简称为熵。当熵中的概率由数据估计（特别是最大似然估计）得到时，所对应的熵称为经验熵。

经验熵公式为

$$H(D) = -\sum_{k=1}^{K}\frac{C_k}{D}\mathrm{lb}\left(\frac{C_k}{D}\right)$$

申请信贷的历史数据如表 9.2 所示。

表 9.2　申请信贷的历史数据

序号	有稳定收入	有房产	审核放贷结果
1	否	否	否
2	否	否	否
3	是	否	是
4	是	是	是
5	否	否	否
6	否	是	是
7	是	否	是
8	是	是	是
9	否	否	否
10	否	是	是
11	是	否	是
12	否	是	是
13	否	否	否
14	否	否	否
15	是	是	是

在 15 个数据中，9 个数据的结果为放贷，6 个数据的结果为不放贷。数据集 D 的经验熵 H(D)为

$$H(D)=-\frac{9}{15}\log_2\frac{9}{15}-\frac{6}{15}\log_2\frac{6}{15}=0.971$$

3. 建立决策树

建立决策树的基本思想是以信息熵为度量构造一棵熵值下降最快的树，到叶子节点处的熵值为零，需要遍历所有特征，选择信息增益最大的特征作为当前的分裂特征，一个特征的信息增益越大，表明属性对样本的熵减少的能力更强，这个属性使得数据由不确定性变成确定性的能力越强。

4. 决策树的学习过程

（1）特征选择

从训练数据的特征中选择一个特征作为当前节点的分裂标准（特征选择的标准不同产生了不同的特征决策树算法）。

（2）决策树生成

根据所选特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长。

（3）剪枝

决策树容易过拟合，需要剪枝来缩小树的结构和规模（包括预剪枝和后剪枝）。

5. 机器学习模块（ex9_12_Learning.py）

程序代码如下：

```
from math import log
import operator
import pickle

"""
函数说明:计算给定数据集的经验熵(香农熵)
Parameters:
    dataset——数据集
Returns:
    shannonEnt——经验熵(香农熵)
"""
def calcShannonEnt(dataSet):
    numEntires = len(dataSet)          # 返回数据集的行数
    labelCounts = {}                   # 保存每个标签出现次数的字典
    for featVec in dataSet:            # 对每组特征向量进行统计
        currentLabel = featVec[-1]     # 提取标签(Label)信息
        if currentLabel not in labelCounts.keys():
                                   #如果标签没有放入统计次数的字典,那么添加进去
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1                 #Label计数
    shannonEnt = 0.0                                   #经验熵(香农熵)
    for key in labelCounts:                            #计算香农熵
        prob = float(labelCounts[key]) / numEntires  #选择该标签的概率
        shannonEnt -= prob * log(prob, 2)              #利用公式计算
```

```
    return shannonEnt                                   # 返回经验熵(香农熵)

"""
函数说明:创建测试数据集
Returns:
    dataset——数据集
    labels——特征标签
"""
def createDataSet():
    dataSet = [[0, 0, 'no'],                            # 表9.2的数据集
               [0, 0, 'no'],
               [1, 0, 'yes'],
               [1, 1, 'yes'],
               [0, 0, 'no'],
               [0, 0, 'no'],
               [0, 0, 'no'],
               [1, 1, 'yes'],
               [0, 1, 'yes'],
               [0, 1, 'yes'],
               [0, 1, 'yes'],
               [0, 1, 'yes'],
               [1, 0, 'yes'],
               [1, 0, 'yes'],
               [0, 0, 'no']]
    labels = ['有稳定收入', '有房产', '审核结果']        # 特征标签
    return dataSet, labels                              # 返回数据集和分类属性

"""
函数说明:按照给定特征划分数据集
Parameters:
    dataset——待划分的数据集
    axis——划分数据集的特征
    value——需要返回的特征的值
Returns:
    无
"""
def splitDataSet(dataSet, axis, value):
    retDataSet = []                                     # 创建返回的数据集列表
    for featVec in dataSet:                             # 遍历数据集
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]             # 去掉axis特征
            reducedFeatVec.extend(featVec[axis+1:])     # 添加到返回的数据集
            retDataSet.append(reducedFeatVec)
    return retDataSet                                   # 返回划分后的数据集
```

```
"""
函数说明:选择最优特征
Parameters:
    dataset——数据集
Returns:
    bestFeature——信息增益最大的(最优)特征的索引值
"""
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1                        # 特征数量
    baseEntropy = calcShannonEnt(dataSet)                    # 计算数据集的香农熵
    bestInfoGain = 0.0                                       # 信息增益
    bestFeature = -1                                         # 最优特征的索引值
    for i in range(numFeatures):                             # 遍历所有特征
        #获取dataSet的第i个所有特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList)                   # 创建set集合{},元素不可重复
        newEntropy = 0.0                                     # 经验条件熵
        for value in uniqueVals:                             # 计算信息增益
            subDataSet = splitDataSet(dataSet,i,value)       # 设置划分后的子集
            prob = len(subDataSet) / float(len(dataSet))     # 计算子集的概率
            newEntropy += prob * calcShannonEnt(subDataSet)# 计算经验条件熵
        infoGain = baseEntropy - newEntropy   # 信息增益
        if (infoGain > bestInfoGain):         # 计算信息增益
            bestInfoGain = infoGain           # 更新信息增益，找到最大的信息增益
            bestFeature = i                   # 记录信息增益最大特征的索引值
    return bestFeature                        # 返回信息增益最大特征的索引值


"""
函数说明:统计classList中出现此处最多的元素(类标签)
Parameters:
    classList——类标签列表
Returns:
    sortedClassCount[0][0]——出现此处最多的元素(类标签)
"""
def majorityCnt(classList):
    classCount = {}
    for vote in classList:                    # 统计classList中每个元素出现的次数
        if vote not in classCount.keys():classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(),
                              key = operator.itemgetter(1),
                              reverse = True)    # 根据字典的值降序排序
    return sortedClassCount[0][0]             # 返回classList中出现次数最多的元素
```

```
"""
函数说明:创建决策树
Parameters:
    dataset——训练数据集
    labels——分类属性标签
    featLabels——存储选择的最优特征标签
Returns:
    myTree——决策树
"""
def createTree(dataSet, labels, featLabels):
    # 取分类标签(是否放贷:yes or no)
    classList = [example[-1] for example in dataSet]
    #如果类别完全相同则停止继续划分
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    #遍历所有特征后返回出现次数最多的类标签
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)  # 选择最优特征
    bestFeatLabel = labels[bestFeat]              # 最优特征的标签
    featLabels.append(bestFeatLabel)
    myTree = {bestFeatLabel:{}}                   # 根据最优特征的标签生成树
    del(labels[bestFeat])                         # 删除已经使用特征标签
    #得到训练集中所有最优特征的属性值
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)                  # 去掉重复的属性值
    for value in uniqueVals:                      # 遍历特征，创建决策树
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet,
        bestFeat, value), labels, featLabels)
    return myTree

"""
函数说明:使用决策树分类
Parameters:
    inputTree——已经生成的决策树
    featLabels——存储选择的最优特征标签
    testVec——测试数据列表，顺序对应最优特征标签
Returns:
    classLabel——分类结果
"""
def classify(inputTree, featLabels, testVec):
    firstStr = next(iter(inputTree))              # 获取决策树节点
    secondDict = inputTree[firstStr]              # 下一个字典
    featIndex = featLabels.index(firstStr)
    for key in secondDict.keys():
```

```
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key], featLabels, testVec)
            else: classLabel = secondDict[key]
    return classLabel
```

6. 显示决策树模块（ex9_12_tree.py）

```
from matplotlib.font_manager import FontProperties
import matplotlib.pyplot as plt
from math import log
import operator
import ex9_12_Learning

"""
函数说明:获取决策树叶子节点的数目
Parameters:
    myTree——决策树
Returns:
    numLeafs——决策树叶子节点的数目
"""
def getNumLeafs(myTree):
    numLeafs = 0                                    # 初始化叶子
    firstStr = next(iter(myTree))                   # 获取节点属性
    secondDict = myTree[firstStr]                   # 获取下一组字典
    for key in secondDict.keys():
        #测试该节点是否为字典，如果不是字典，代表此节点为叶子节点
        if type(secondDict[key]).__name__=='dict':
            numLeafs += getNumLeafs(secondDict[key])
        else:   numLeafs +=1
    return numLeafs

"""
函数说明:获取决策树的层数
Parameters:
    myTree——决策树
Returns:
    maxDepth——决策树的层数
"""
def getTreeDepth(myTree):
    maxDepth = 0                                    # 初始化决策树深度
    firstStr = next(iter(myTree))
    secondDict = myTree[firstStr]                   # 获取下一个字典
    for key in secondDict.keys():
#测试该节点是否为字典，如果不是字典，代表此节点为叶子节点
if type(secondDict[key]).__name__=='dict':
```

```
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth    # 更新层数
    return maxDepth

"""
函数说明:绘制节点
Parameters:
    nodeTxt——节点名
    centerPt——文本位置
    parentPt——标注的箭头位置
    nodeType——节点格式
"""
def plotNode(nodeTxt, centerPt, parentPt, nodeType):
    arrow_args = dict(arrowstyle="<-")                   # 定义箭头格式
    # 设置简体汉字
    font = FontProperties(fname=r"c:\windows\fonts\simsun.ttc", size=14)
    createPlot.ax1.annotate(nodeTxt, xy=parentPt,
        xycoords='axes fraction',                        #绘制节点
        xytext=centerPt, textcoords='axes fraction',
        va="center", ha="center", bbox=nodeType,
        arrowprops=arrow_args, FontProperties=font)

"""
函数说明:标注有向边属性值
Parameters:
    cntrPt、parentPt——用于计算标注位置
    txtString——标注的内容
"""
def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]           # 计算标注位置
    yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center",
    rotation=30)

"""
函数说明:绘制决策树
Parameters:
    myTree——决策树(字典)
    parentPt——标注的内容
    nodeTxt——节点名
"""
def plotTree(myTree, parentPt, nodeTxt):
    decisionNode = dict(boxstyle="sawtooth", fc="0.8")  # 设置节点格式
    leafNode = dict(boxstyle="round4", fc="0.8")        # 设置叶子节点格式
```

```
    numLeafs = getNumLeafs(myTree)   # 获取决策树叶子节点数目，决定了树的宽度
    depth = getTreeDepth(myTree)     # 获取决策树层数
    firstStr = next(iter(myTree))    #下个字典
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs))/2.0/plotTree.totalW,
plotTree.yOff)                       # 中心位置
    plotMidText(cntrPt, parentPt, nodeTxt)              # 标注有向边属性值
    plotNode(firstStr, cntrPt, parentPt, decisionNode)  # 绘制节点
    secondDict = myTree[firstStr]    # 下一个字典，也就是继续绘制子节点
    plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD # y偏移

    for key in secondDict.keys():
      # 测试该节点是否为字典，如果不是字典，代表此节点为叶子节点
      if type(secondDict[key]).__name__=='dict':
         plotTree(secondDict[key],cntrPt,str(key))
                                                # 不是叶子节点，递归调用继续绘制
      else:
        # 如果是叶子节点，绘制叶子节点，并标注有向边属性值
        plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
        plotNode(secondDict[key],(plotTree.xOff,plotTree.yOff),cntrPt,leafNode)
        plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD

"""
函数说明:创建绘制面板
Parameters:
    inTree——决策树(字典)
"""
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')  #创建fig
    fig.clf()       #清空fig
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)# 去掉x、y轴
    plotTree.totalW = float(getNumLeafs(inTree))     # 获取决策树叶子节点数目
    plotTree.totalD = float(getTreeDepth(inTree))    # 获取决策树层数
    plotTree.xOff = -0.5/plotTree.totalW; plotTree.yOff = 1.0;# x偏移
    plotTree(inTree, (0.5,1.0), '')                  # 绘制决策树
    plt.show()                                       # 显示绘制结果

def main():
    dataSet, labels = ex9_12_Learning.createDataSet()
    featLabels = []
    myTree = ex9_12_Learning.createTree(dataSet, labels, featLabels)
    print(myTree)
    createPlot(myTree)
```

```
if __name__ == '__main__':
    main()
```

程序运行结果如图 9.18 所示。

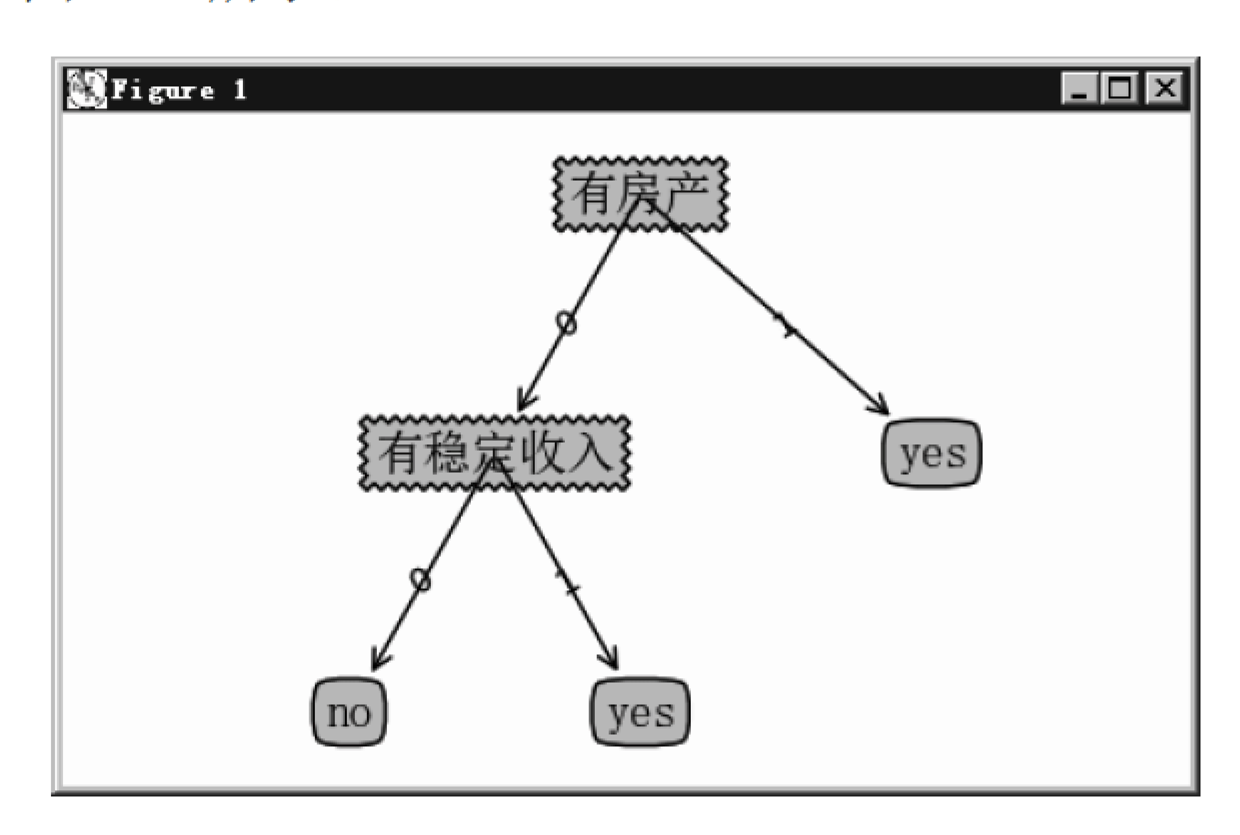


图 9.18　显示决策树

7. 界面程序（ex9_12.py）

程序代码如下：

```
from tkinter import *
from tkinter import ttk
import ex9_12_tree
import ex9_12_Learning

win = Tk()
win.geometry('450x116')
win.title('信贷审核')

dataSet, labels = ex9_12_Learning.createDataSet()
featLabels = []
myTree = ex9_12_Learning.createTree(dataSet, labels, featLabels)

"""
函数说明：审核按钮事件
Parameters:
    txt1——选择框输入的收入状况
    txt2——选择框输入的房产状况
    txt3——显示决策树的判断结果
Returns:
    无
"""
def mClick():
  txt1 = nChosen1.get()
  txt2 = nChosen2.get()
  if txt1 == '有稳定收入':
```

```
        t1 = 1
    else:
        t1 = 0
    if txt2 == '有房产':
        t2 = 1
    else:
        t2 = 0
    testVec = [t1,t2]    # 测试数据,用0和1表示
    result = ex9_12_Learning.classify(myTree, featLabels, testVec)
    if result == 'yes':
        txt3.set("审核结果：放贷")
    if result == 'no':
        txt3.set("审核结果：不放贷")

"""
函数说明：显示决策树按钮事件
"""
def mClick2():
    ex9_12_tree.main()

# 创建几个组件元素
txt1=StringVar()
txt2=StringVar()
txt3=StringVar()
txt3.set("决策树判断放贷结果")
lab1=Label(win, text="请选择收入状况：",font=('宋体','16'))
lab2=Label(win, text="请选择房产状况：",font=('宋体','16'))
lab3=Label(win,textvariable=txt3,relief='ridge',width=30,font=('宋体', '16'))
button = Button(win, text='信贷审核', command=mClick,font=('宋体','16'))
button2 = Button(win, text='显示决策树', command=mClick2,font=('宋体','14'))

# 创建下拉列表
nChosen1=ttk.Combobox(win,width=12,textvariable=txt1,font=('宋体','16'))
nChosen1['values'] = ('','有稳定收入', '无稳定收入')
nChosen1.current(0)  # 设置下拉列表默认值
nChosen2=ttk.Combobox(win,width=12,textvariable=txt2,font=('宋体','16'))
nChosen2['values'] = ('','有房产', '无房产')
nChosen2.current(0)  # 设置下拉列表默认值

# 界面布局设置
lab1.grid(row=0,column=0)
lab2.grid(row=1,column=0)
nChosen1.grid(row=0,column=1)
nChosen2.grid(row=1,column=1)
lab3.grid(row=2,column=0,columnspan=2)
```

```
button.grid(row=2,column=2)
button2.grid(row=0,column=2)

win.mainloop()
```

程序运行结果如图 9.19 所示。



图 9.19　应用机器学习得出信贷审核结果

图书资源支持

感谢您一直以来对清华版图书的支持和爱护。为了配合本书的使用，本书提供配套的资源，有需求的读者请扫描下方的“书圈”微信公众号二维码，在图书专区下载，也可以拨打电话或发送电子邮件咨询。

如果您在使用本书的过程中遇到了什么问题，或者有相关图书出版计划，也请您发邮件告诉我们，以便我们更好地为您服务。

我们的联系方式：

地　　址：北京海淀区双清路学研大厦 A 座 707

邮　　编：100084

电　　话：010-62770175-4604

资源下载：http://www.tup.com.cn

电子邮件：weijj@tup.tsinghua.edu.cn

QQ：883604(请写明您的单位和姓名)

资源下载、样书申请



书圈

用微信扫一扫右边的二维码，即可关注清华大学出版社公众号“书圈”。